Numéro d'ORDRE: 03/2019-D/MT

RÉPUBLIQUE ALGÉRIÈNNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMÉDIÈNE
FACULTÉ DES MATHÉMATIQUES

THÈSE de DOCTORAT en SCIENCES

Présentée pour l'obtention du grade de **DOCTEUR**

En: **MATHÉMATIQUES**

Spécialité: **Recherche Opérationnelle: Mathématiques de Gestion**

Par **AHMIA Ibtissam**

Titre

Une nouvelle métaheuristique pour les problèmes d'optimisation combinatoire: La Monarchie Métaheuristique

Soutenue le 27/04/2019, devant le jury composé de:

M^r	OUAFI Rachid	Professeur	à l'USTHB	Président
M^r	AÏDER Méziane	Professeur	à l'USTHB	Directeur de Thèse
M^r	AÏDENE Mohamed	Professeur	à l'UMM/Tizi-Ouzou	Examinateur
M^r	OUKACHA Brahim	Professeur	à l'UMM/Tizi-Ouzou	Examinateur

Résumé

Dans cette thèse nous présentons une nouvelle métaheuristique que nous avons développé intitulée "la Monarchie Métaheuristique". Notre méthode a été inspirée du système de gouvernement la monarchie. Contrairement à beaucoup d'autres métaheuristiques, cette métaheuristique est facile à implémenter et ne nécessite pas beaucoup de paramètres. Cela la rend applicable à un large éventail de problèmes d'optimisation combinatoire. Pour évaluer l'efficacité de la métaheuristique proposée, nous avons examiné ses performances sur le problème de voyageur de commerce (TSP) en utilisant des instances de la bibliothèque d'instances en ligne TSPLIB pour le TSP. Les résultats expérimentaux indiquent que la Monarchie Métaheuristique est en concurrence avec les autres méthodes existantes dans la littérature pour trouver des solutions approchées.
Mots clés : Métaheuristiques, la monarchie métaheurstique, les méthodes hybrides, l'optimisation combinatoire.

Abstract

In this thesis, we introduce a novel metaheuristic optimization algorithm named the Monarchy Metaheuristic (MN). Our proposed metaheuristic is inspired from the monarchy government system. Unlike many other metaheuristics, this metaheuristic is easy to be implemented and does not need a lot of parameters. This makes it applicable to a wide range of combinatorial optimization problems. To evaluate the efficiency of the proposed algorithm, we examined it on the Traveling Salesman Problem (TSP) using some benchmark from TSPLIB online library of instances for the TSP. The experimental results indicate that the monarchy metaheuristic is competitive with the other methods that exist in the literature for finding approximate solutions.

Key words: Metaheuristics, the monarchy metaheuristic, hybrid-methods, combinatorial optimization.

Je dédie cette thèse,

À ma très chère mère,
pour ton dévouement et tes sacrifices, tu es et tu resteras toujours ma source d'inspiration. Tes prières ont illuminé ma voie et m'ont été d'un grand secours pour atteindre mes buts dans la vie. Tu as toujours été à mes cotés pour me soutenir, m'encourager et m'épauler. Tu es et tu resteras toujours mon professeur, mon réconfort, et ma meilleure amie. Tout ce que je suis et ce que j'espère devenir, je le dois à ma très chère mère.
À mon très cher père,
Ton soutien permanent et inconditionnel a renforcé mon esprit fonceur et ma motivation pour réaliser mes rêves. Cher père, tu es et tu restera pour toujours mon héros, mon confident, mon conseiller, et tout ce que tu m'as appris, restera pour moi un héritage precieurx pour toute ma vie.
Tous les mots ne pourraient témoigner de ma gratitude, et ma reconnaissante à vous chers parents. J'espère que vous trouverez dans cette thèse le fruit de vos efforts. Je vous souhaite une longue vie de bonheur inchaALLAH.
À ma fille Besma, mon ange et la lumière qui illumine ma vie.
À mon frère Imam.
À mon mari.
Et à mes étudiants,
pour vous prouver que publier un article scientifique nécessite une volonté de réussir, des efforts personnnels et un simple Laptop :) pas plus, ça ne dépend pas du lieu (pays) du travail. Je suis fière d'avoir réalisé mon travail en Algerie.

REMERCIEMENTS

Tout d'abord, je remercie ALLAH, de m'avoir doté de volonté et du courage pour la réalisation de ce travail et pour toutes ces bénédictions qu'il m'a offertes.

Je remercie mes parents Mr AHMIA Fatah et Mme BOUDER Nadia qui, grâce à leurs sacrifices et dévouement, j'ai réussi à achever cette thèse, ils ont cru en moi plus que je ne le faisais moi-même. Ils m'ont soutenu et aidé tout au long de la réalisation de ce travail de recherche et m'ont mis en place les conditions optimales pour pouvoir se concentrer sur mon travail.

Je remercie mon directeur de thèse le professeur AÏDER Meziène, je lui serais toujours reconnaissante de m'avoir encourager à croire en mes idées et d'aller jusqu'au bout jusqu'à les voir se concrétiser. Je le remercie pour ses conseils pertinents, sa compréhension, sa disponibilité et ses encouragements.

Mes remerciements vont ensuite aux professeurs qui m'ont honoré en acceptant d'être membre de mon jury de thèse et qui ont pris de leur temps pour évaluer mon travail et donner leur avis :
Mr OUAFI Rachid professeur à l'Université des Sciences et Technologie Houari boumedien le président du jury.
Mr AÏDENE Mohamed et OUKACHA Brahim tous deux professeurs à l' Université Mouloud Mammeri de Tizi Ouzou, examinateurs de cette thèse.

Enfin, je remercie mon frère de m'avoir toujours embêter avec sa question "Quand est ce que tu termine ton doctorat ?". Je lui dit elhamdouLAH, je l'ai enfin terminé. Un travail de

recherche est un processus souvent stochastique, il n’est pas deterministe, on ne peut pas prévoir avec exactitude ses résultats et la date de son achèvement.

TABLE DES MATIÈRES

Remerciements **I**

Table des matières **IV**

Table des figures **X**

Introduction **1**

1 Optimisation Combinatoire **3**

1.1 Exemple de problèmes d'Optimisation Combinatoire 4

1.1.1 Le problème du voyageur de commerce 4

1.1.2 Le problème de sac à dos binaire 5

1.1.3 Le problème d'affectation 6

1.1.4 Le problème de tournées de véhicules 6

1.2 Complexité des problèmes Combinatoires 8

1.2.1 Niveaux de complexités 10

1.2.2 Classes de Complexité 11

1.3 Methodes de résolution des problèmes d'Optimisation Combinatoire 13

1.3.1 Methodes de résolution exactes 14

1.3.1.1 Branch and Bound 15

1.3.1.2 Branch and Cut 16

1.3.1.3 Branch and Price 16

1.3.2 Methodes de résolution approchées 17

1.3.2.1 Les heuristiques constructives [1] 17

1.3.2.2 La recherche locale [1] 18

1.3.2.3 Les métaheuristiques 19

2 Les Métaheuristiques et Méthodes Hybrides 20

2.1 Classifications des Métaheuristiques 22
2.2 Métaheuristiques basées sur des métaphores 23
2.3 Concepts importants des métaheuristiques 25
2.4 Les métaheuristiques à solution unique 27
2.4.1 La méthode de descente (Hill Climbing) 28
2.4.2 La Recherche Tabou 29
2.4.3 Le Recuit Simulé 30
2.4.4 La méthode GRASP 31
2.4.5 La recherche à voisinage variable 32
2.5 Les métaheuristiques à population de solutions 33
2.5.1 L'algorithme génétique 34
2.5.2 L'algorithme à essaims de particules 35
2.6 L'hybridation des métaheuristiques 35
2.6.1 Classification hiérarchique des méthodes hybrides 36
2.6.1.1 L'hybridation relais de bas niveau (BNR) 37
2.6.1.2 L'hybridation co-évolutionnaire de bas niveau (BNC) 37
2.6.1.3 L'hybridation relais de haut niveau (HNR) 37
2.6.1.4 L'hybridation co-évolutionnaire de haut niveau (HCH) 38

3 Définition d'une nouvelle approche Métaheuristique : La Monarchie Metaheuristique 39

3.1 Définitions nécessaires pour comprendre la méthode proposée 39
3.1.1 Monarchie héréditaire 39
3.1.2 Dynastie 40
3.1.3 L'ordre de succession 40
3.2 Description de la monarchie metaheuristique 40
3.2.1 Trouver le premièr Roi 42
3.2.2 Creation de la liste L_1 42
3.2.2.1 L'opérateur de croisement OX (Order Crossover (OX)) [2] . 42
3.2.2.2 L'opérateur de croisement à un point (Single Point Crossover) [2,3] 43
3.2.2.3 L'opérateur de croisement à deux points (Two Point Crossover) [2,3] 44

3.2.2.4 L'opérateur de croisement uniforme (Uniform Crossover) (UX) [2] . . . 44
3.2.3 Creation de la liste L_2 . . . 44
3.2.4 L'inversion adjacente (Adjacent swap) [4] . . . 46
3.2.5 L'inversion (Swap) [4] . . . 46
3.2.6 Voisinage 2-opt . . . 47
3.2.7 Sélection du prochain Roi . . . 47
3.2.7.1 Primogéniture . . . 47
3.2.7.2 La succession adelphique . . . 47
3.2.7.3 Sélection semi-élective . . . 48
3.2.7.4 L'intrus . . . 48
3.2.8 Le framework de la Monarchie Métaheuristique . . . 48
3.2.9 le paramètre de la monarchie métaheuristique . . . 48
3.2.10 Et si la prochaine solution Roi n'apporte pas d'amélioration ? . . . 51

4 Une première variante de la Monarchie Metaheuristique appliquée pour la resolution du problème de voyageur de commerce 52
4.1 Le problème traité : Le problème du voyageur de commerce (TSP) . . . 52
4.2 Application de la Monarchie Metaheuristique au problème TSP . . . 53
4.2.1 Création de la première solution Roi . . . 53
4.2.2 Creation de la liste L_1 . . . 53
4.2.3 Creation de la liste L_2 . . . 54
4.2.3.1 Le voisinage à un point . . . 55
4.2.3.2 Le voisinage à deux points . . . 55
4.2.3.3 Le voisinage 2-opt . . . 56
4.2.4 Selection du prochain Roi . . . 57
4.2.4.1 Primogéniture . . . 57
4.2.4.2 Sélection Adelphique . . . 57
4.2.4.3 Sélection semi-élective . . . 57
4.2.4.4 L'intrus . . . 58
4.3 Résultats expérimentaux et discussion . . . 59

5 Adaptation de la Monarchie Metaheuristique (MN) à la résolution de d'autres problèmes d'OC 68
5.1 Adaptation de MN au problème du sac à dos binaire . . . 68
5.1.1 Création de la première solution Roi . . . 69

5.2 Création de la liste L_1 69
5.2.1 Le croisement uniforme [5] 69
5.2.2 le croisement 1-BX [5] 70
5.2.3 Le croisement RRC [6] [5] 71
5.3 Creation de la liste L_2 71
5.3.1 Voisinage par complémentation (remplacement) : 71
5.4 Stratégies utilisées dans le cas d'irréalisabilité d'une solution 72
5.5 Qu'elle méthode de résolution utiliser pour créer la prochaine solution Roi quand le mode de selection du prochain Roi est l'intrus : 72
5.6 Adaptation de MN au problème VRP avec contrainte de capacité 72
5.6.1 Création de la première solution Roi 73
5.6.2 Création de la liste L_1 74
5.6.3 Création de la liste L_2 74
5.6.3.1 Les voisinages qui opérent sur une même tournée (intra route operators : 74
5.6.3.2 Les voisinages qui opérent sur deux tournées) (inter route operators : 75
5.6.4 Qu'elle méthode de résolution utiliser pour créer la prochaine solution Roi quand le mode de selection du prochain est l'intrus : 75

Conclusion **76**

Bibliographie **78**

LIST OF ALGORITHMS

1 **L'algorithme de branch-and-bound** 16

2 **Heuristique constructive** [1] 18

3 **Recherche locale** [1] 19

4 **Cadre générique d'une Métaheuristique** [7] 27

5 **Pseudo code de la méthode de descente** 29

6 **Pseudo code de l'Algorithme de Recherche Tabou** 29

7 **Pseudo code de l'Algorithme du Recuit Simulé** 30

8 **Pseudo code de l'Algorithme de GRASP** 31

9 **Pseudo code de l'algorithm de la recherche à voisinage variable** [8] . . . 33

10 **Pseudo code de l'algorithm génétique** 34

11 **Pseudo code de l'algorithme à essaims particulaires** 35

12 **Pseudocode de la Monarchie Métaheuristique** 49

13 **Algorithme Glouton** 54

14 **Le voisinage à un point (↓ Tournée : une tournée, K : entier positif, pos : entier positif, ↑ Voisins : une liste de tournées)** 55

15 **Le voisinage à deux points (↓ Tournée : une tournée, K : entier positif, pos : entier positif, ↑ Voisins : une liste de tournées)** 56

16 **Le voisinage 2-Opt (↓ Tour : une tournée, pos1 : entier positif, ↑ Voisins : une liste de tournées** 56

17 **Procedure Primogéniture (↓ L_1 : une liste de tournées, ↑ T : une tournée)** 57

18 **Procedure Adelphique (↓ L_2 : une liste de tournées, ↑ T : une tournée)** . 57

19 **Procedure Semi-élective (↓ L_1, L_2 : deux listes de tournées, ↑ T : une tournée)** 58

20 **Procedure Intrus (↑ $Solution$: une tournée)** 58

21 **Création** $1^{ère}$ **solution Roi pour 0-1 KP** 69
22 **Heuristique de construction de la première solution Roi (cas : CVRP)** . . 74

TABLE DES FIGURES

1.1 Diagramme d'Euler pour représenter les classes de problèmes P, NP, NP-complete, and NP-hard. 14
1.2 Une possible hièrarchie des méthodes d'optimisation [9] 17

2.1 Métaphore utilisées par les chercheurs pour le dévelopement de nouvelles métaheuristiques. [10] 25
2.2 Intensification [9] 26
2.3 Diversification [9] 26
2.4 Illustration du fonctionnement d'une métaheuristique à base de solution unique (S-métaheuristique) [11]. 28
2.5 Illustration du fonctionnement du VNS [11]. 32
2.6 Illustration du fonctionnement d'une métaheuristique à base de population (P-métaheuristique) [11]. 34
2.7 Les différents niveaux d'hybridation 36
2.8 Les différents modes d'hybridation 37

3.1 Exemple de dynastie de la monarchie de la grande bretagne. 41
3.2 Illustration du Order Crossover (OX). 43
3.3 Illustration du croisement en un point. 43
3.4 Illustration du croisement en deux points. 44
3.5 Illustration du croisement Uniforme (UX). 45
3.6 L'espace des permutations de taille 4 connectées selon le voisinage adjacent swap [4] 46
3.7 Organigramme de la Monarchie Métaheuristique 50

4.1 Comparaison des valeurs de la fonction objectif pour l'instance Eil51 de TSP (distances entières). 59
4.2 Comparaison des valeurs de la fonction objectif pour l'instance Pr76 de TSP (distances entières). 59
4.3 Comparaison des valeurs de la fonction objectif pour Berlin52, Eil76, St70 instances de TSP (distances entières). 62
4.4 Comparaison des valeurs de la fonction objectif pour Kroa100, Eil101 instances de TSP (distances entières). 62
4.5 Comparaison des RE% de MN2 avec ceux de ACO et HA (distances réelles). . 64
4.6 Comparaison des RE% de MN1, MN2 avec ceux de ACO, ABC, HA (distances entières). 64
4.7 Comparaison des valeurs de la fonction objectif pour les instances Eil51, St70, Eil76, Eil101 de TSP (distances réelles). 64
4.8 Comparaison des valeurs de la fonction objectif pour les instances Pr76, Kroa100, Berlin52 de TSP (distances réelles). 64
4.9 Comparaison des temps de calcul de ACO, HA (ACO+ABC) avec celui de MN2 (distances réelles). 65
4.10 Courbe de convergence des meilleurs résultats pour MN_2 pour l'instance Eil51 en démarrant par une solution trouvée par une heuristique gloutonne. 65
4.11 Courbe de convergence des meilleurs résultats pour MN_2 pour l'instance Pr76 en démarrant par une solution trouvée par une heuristique gloutonne. 65
4.12 Comparaison des courbes de convergence entre MN_2 et GA, les deux ont démarré par des solutions générées aléatoirement pour l'instance Eil51. 66
4.13 Comparaison des courbes de convergence entre MN_2 et GA, les deux ont démarré par des solutions générées aléatoirement pour l'instance Pr76. 66

5.1 Exemple d'application de croisement à un point 70
5.2 Exemple d'application de croisement à k points (pour cet exemple on a 4 points de croisement) 70
5.3 Exemple d'application de croisement uniforme 70

L'homme scientifique ne vise pas à obtenir un résultat immédiat. Il ne s'attend pas à ce que ses idées avancées soient facilement adoptées. Son travail est comme celui du planteur pour l'avenir. Son devoir est de jeter les bases pour ceux qui sont à venir et de montrer la voie.

Nikola Tesla

INTRODUCTION

L'optimisation est essentiellement un outil d'aide à la décision au sein de l'entreprise, et aussi pour les individus. En effet, le problème d'optimiser se retrouve dans tous les domaines de l'activité économique, politique, scientifique et sociale. Car toute personne dans n'importe quel secteur cherche à trouver la répartition optimale des moyens limités qu'elle possède.
La citation des auteurs Wild, Beightler, et Phillips dans leur livre Foundations of optimization en 1979 [12] à propos de l'optimisation permet d'avoir une idée sur le but de cette branche d'études : "Le désir humain de perfection trouve son expression dans la théorie de l'optimisation. Elle étudie comment décrire et atteindre ce qui est meilleur, une fois que l'on connaît comment mesurer et modifer ce qui est bon et ce qui est mauvais . . . la théorie de l'optimisation comprend l'étude quantitative des optimums et les méthodes pour les trouver."
Mathématiquement parlant, l'optimisation signifie maximiser où minimiser une fonction de plusieurs variables en respectant un ensemble de contraintes. La nature du domaine d'appartenance des variables spécifie la nature de l'optimisation qu'on cherche à faire, pour l'optimisation discrète ou combinatoire par exemple le domaine est discret. Et c'est les problèmes qui appartiennent à la classe d'optimisation combinatoire qui nous interessent.
L'optimisation combinatoire est liée à la recherche opérationnelle, à la théorie des algorithmes, et la théorie de la complexité informatique. L'optimisation combinatoire fait toujours l'objet de recherches intensives et s'applique aujourd'hui à de nombreux domaines : finance (minimisation de coût, maximisation de projet), transport (planification), surveillance (détection de dysfonctionnement, prévention), biologie (analyse de données, modélisation), etc. Et c'est pour cette raison que nous devons être capables de résoudre ce type de problèmes de manière efficace. Malheureusement, bon nombre de ces problèmes s'avèrent être de complexity NP-difficiles, c'est-à-dire qu'il est souvent impossible de les résoudre de ma-

nière exacte en un temps raisonnable. Cependant, dans la pratique, on n'a généralement pas besoin d'une solution exacte du problème. Dans ce cas, on peut utiliser un algorithme heuristique ou une métaheuristique qui donne une solution approchée dans un temps satisfaisant. Dans cette thèse une nouvelle méthode approchée (métaheuristique) que nous avons dévelopée destinée à la résolution des problèmes d'optimisation combinatoire sera présentée.

Cette thèse est organisée comme suit :
Dans le premier chapitre, nous présentons une vue globale sur la classe des problèmes d'optimisation combinatoire, nous définissons ce qui'est un problème d'optimisation combinatoire, des exemples de problèmes qui appartienent à cette classe, Nous détaillons ensuite les notions de complexité relative à ce type de problèmes, et nous présentons quelques méthodes de résolution exactes et approchées (heuristiques ou métaheuristiques).

Dans le chapitre deux, nous nous orientant exclusivement vers les métaheuristiques qu'elles soient à base de solution unique ou à population de solutions. En effet, dans cette thèse, nous ne travaillons pas sur les méthodes dites exactes. Une fois les différents types de métaheuristiques présentés, nous définissons ce qui'est l'hybridation des méthodes et la classification des méthodes hybrides.

Le chapitre trois présente notre contibution, l'approche métaheuristique la monarchy métaheuristique que nous avons réalisé pour la résolution des problèmes d'optimisation combinatoire, le cadre général de la méthode est présenté dans ce chapitre ainsi qu'une déscription détaillée du fonctionnement de cette dernière.

Dans le chapitre quatre, une première variante de notre métaheuristique est présentée, application de la méthode sur le problème connu du voyageur de commerce, ses resultats expérimentaux sont présentés ainsi qu' une comparaison de ses résultats avec d'autres méthodes qui existent dans la littérature.

Le chapitre trois et quatre ont été rédigé à base de notre article [13] publié dans la revue "Turkish Journal of Electrical Engineering and Computer Sciences".

Enfin, cette thèse s'achève par une conclusion générale, où nous récapitulons nos contributions et proposons des perspectives, sur la base des travaux effectués.

I don't care that they stole my idea. I care that they don't have any of their own.

Nikola Tesla

CHAPITRE 1 OPTIMISATION COMBINATOIRE

Beaucoup de problèmes d'une importance pratique et théorique se modélisent comme des problèmes d'optimisation. En effet, nous faisons appel aux concept d'optimisation dans notre vie quotidienne sans forcement en être conscient, en cherchant le trajet le plus court en temps ou en distance pour arriver à une distination donnée, en cherchant à trouver la meilleure façon pour ranger les affaires, en cherchant à augmenter sa productivité au travail, en essayant de bien gérer son salaire pour qu'il tienne jusqu'à la fin du mois, la liste des situations qui font appel au concept d'optimisation de manière directe ou indirecte est très longue : meilleur trajet, meilleure organisation, meilleure gestion,. . . du fait que toute ressource (temps, espace, energie, argent,. . .) est limitée, le concept d'optimisation s'impose naturellement.
Un problème d'optimisation consiste donc à chercher la meilleure configuration d'un ensemble de variables pour réaliser certains buts. Les problèmes d'optimisation se devisent en deux classes selon la nature des variables utilisées, on parle alors d'optimisation continu si les variables sont réelles et d'optimisation discrete si les variables sont discretes, parmi les problèmes d'optimisation discrète on peut distinguer les problèmes d'optimisation combinatoire.
Avant de définir ce qu'est un problème d'optimisation combinatoire, nous allons tout d'abord définir ce qu'est un problème combinatoire. De façon informelle, un problème combinatoire correspond à une situation dans laquelle un nombre, généralement très important, de configurations ou possibilités sont envisageable, à priori, mais seules quelques une d'entre elles satisfont les propriétés recherchées.

Définition 1 *Un Problème d'Optimisation Combinatoire $P = (S, f)$ peut être défini par :*

- *Un ensemble de variables $X = \{x_1, x_2, \ldots, x_n\}$.*
- *Une liste de domaines variés $\{D_1, D_2, \ldots, D_n\}$.*
- *Contraintes sur les variables.*
- *Une fonction objectif f à minimiser où $f : \quad D_1 \times \ldots \times D_n \mapsto \mathbb{R}^+$*
- *L'ensemble de toutes les affectations réalisables est $S = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\}/v_i \in D_i\}$, s satisfait toutes les contraintes. S est appelé l'espace de recherche ou espace des solutions, S est un ensemble fini.*

Un problème d'optimisation combinatoire correspond donc à la donnée d'un ensemble S de possibilités, configurations, ou états envisageables et d'une fonction f permettant de mesurer la qualité des éléments de S. Résoudre un tel problème consiste à trouver, parmi les éléments de S l'élément minimisant ou maximisant la valeur de f. La dificulté du problème vient du fait que l'ensemble S est souvent de taille très importante.
Dans cette thèse, nous nous intéressons aux problèmes d'optimisation combinatoire et nous citerons quelques notions qui concerne cette classe de problèmes.

1.1 Exemple de problèmes d'Optimisation Combinatoire

Comme exemple de problème d'optimisation combinatoire on peut citer le TSP (Traveling salesman problem) le problème de voyageur de commerce qui est l'exemple le plus connu de problèmes d'optimisation combinatoires. Autres exemples sont le problème affectation, de planification, et le VRP (vehicle routing problem) problème de tournées de véhicules.

1.1.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce (TSP) a été étudié au 18 ème siècle par un mathématicien d'Irlande nommé William Rowam Hamilton et par le mathématicien britannique nommé Thomas Penyngton Kirkman. Discussion détaillée sur le travail de Hamilton et Kirkman peut être vu à partir du livre intitulé Graph Theory [14]. C'est un problème relevant de la classe NP, il est même NP-complet. Sous sa forme la plus classique, son énoncé est le suivant : " Un voyageur de commerce doit visiter une et une seule fois un nombre fini de villes

et revenir à son point d'origine. Trouvez l'ordre de visite des villes qui minimise la distance totale parcourue par le voyageur ". Le problème TSP se formule mathématiquement comme suit :

$$
(TSP)\begin{cases}
\text{Min } Z(X) = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
& \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in N \\
& \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in N \\
& \sum_{i \in T}\sum_{j \in \overline{T}} x_{ij} \geq 1 \quad \forall T \subset N, \overline{T} = N \backslash T \\
& x_{ij} \in \{0,1\} \quad \forall i,j \in N
\end{cases}
$$

Où $N = \{1, 2, \dots, n\}$ l'ensemble des villes à visiter.

$$
x_{ij} = \begin{cases} 1 & \text{on effectue le trajet allant de i à j,} \\ 0 & \text{sinon} \end{cases}
$$

1.1.2 Le problème de sac à dos binaire

Le problème de sac à dos, noté également KP (en anglais, Knapsack Problem) est l'un des 21 premiers problèmes NP-complets présentés dans l'article de Richard Karp [15]. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec un ensemble d'objets ayant chacun un poids et un profit. Les objets mis dans le sac à dos doivent maximiser le profit total, sans dépasser le poids maximum.

Soit un ensemble de n éléments et une ressource disponible en quantité limitée b. Pour $j = \{1, \dots, n\}$, on note p_j le profit associé à la sélection de l'élément j et on note a_j la quantité de ressource que nécessite l'élément j, s'il est sélectionné. Les coefficients p_j et a_j prennent des valeurs positives pour tout $j = \{1, \dots, n\}$. Le problème du sac-à-dos consiste à choisir un sous-ensemble des n éléments qui maximise le profit total obtenu, en respectant la quantité de ressource disponible.

Une variable de selection x_j est associée à chaque element j qui est égale à 1 si j est sélectionné et à 0 sinon. Le profit total s'écrit $\sum_{i=1}^{n} p_j.x_j$ et la quantité totale de ressource utilisée qui doit etre inferieur à b (la capacité maximal du sac), sera représentée par la contrainte

$\sum_{j=1}^{n} a_j.x_j \leq b$. Le problème de sac à dos binaire se modélise donc sous la forme :

$$\begin{cases} Max \sum_{j=1}^{n} p_j.x_j \\ s.c \quad \sum_{j=1}^{n} a_j.x_j \leq b \\ x_j \in \{0,1\} \quad \forall j = \{1,\ldots,n\} \end{cases}$$

1.1.3 Le problème d'affectation

C'est un problème relevant de la classe P bien évidement parce que l'algorithme de Khun connu sous le nom de la méthode Hongroise [16] représente un algorithme polynomial pour sa résolution.

Le problème d'affectation consiste à affecter n personnes à n tâches. Dans le cas mono-objectif, un coût est associé à l'affectation d'une personne à une tâche et l'idée est de déterminer la permutation des n objets ayant le coût total minimal. Soit i l'indice des personnes, j l'indice des objets et c_{ij} le coût de l'affectation de i à j. Ce problème se formule mathématiquement comme suit :

$$(AP)\begin{cases} \text{Min } Z(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \sum_{i=1}^{n} x_{ij} = 1 \quad j = \{1,\ldots,n\} \\ \sum_{j=1}^{n} x_{ij} = 1 \quad i = \{1,\ldots,n\} \\ x_{ij} \in \{0,1\} \quad i,j = \{1,\ldots,n\} \end{cases}$$

Où c_{ij} sont des entiers positifs et $x = (x_{11},\ldots,x_{nn})$ est la matrice des variables de decision telle que $x_{ij} = 1$ si la personne i est affecté à l'objet j et est égale à 0 sinon.

1.1.4 Le problème de tournées de véhicules

Le problème de tournées de véhicules (VRP) est l'un des problèmes les plus célèbres et les plus étudiés en optimisation combinatoire depuis sa formulation par Dantzig et Ramser en 1959 [17], en raison de sa grande présence dans les applications réelles. C'est une généralisation du problème du voyageur de commerce (TSP), c'est pourquoi il est également un

problème NP-difficile. Il peut être décrit comme un problème de conception de tournées de plusieurs véhicules, à moindre coût, d'un dépôt à un ensemble de clients géographiquement dispersés, de telle sorte de livrer tous les clients et de respecter les contraintes de capacité des véhicules.

Il existe une grande variété de surveys sur le VRP dans la littérature, on peut citer celui de Raff (1983) [18], Christofides [19], Laporte (1990) [20], Desrochers, Lenstra and Savelsbergh (1990) [21], Toth et al. [22].

Plus formellement, le problème de tournées de véhicules est décrit par un graphe $G = (L, E)$ composé des ensembles L et E, l'ensemble $L = \{0, \ldots, N\}$ où 0 est le dépot où stationnent K véhicules de capacité Q_k et $L \setminus \{0\}$ sont N clients qui doivent être servies, caractérisés par une demande de produit non négative q_i et l'ensemble $E = \{(i, j) | i, j \in N, i \neq j\}$ représente l'ensemble des arêtes reliant les sommets.

La matrice $C = (c_{ij})$ est définie sur E, où c_{ij} est le côut sur l'arête, dans un certain contexte il peut être interprété comme un côut de transport, durée de voyage, ou distance de voyage d_{ij}. Généralement le problème VRP est traité comme symétrique, oû $c_{ij} = c_{ji}$. En pratique, la matrice des coûts est asymétrique et doit être calculée à partir de données géographiques à l'aide des algorithmes de plus court chemin. Dans la formulation donnée ci dessous D_k représente la distance maximale autorisée pour le véhicule k.

N.Christofides (1985) [23] présente trois formulations différentes au début des années 1980 pour le problème de tournées de véhicules, la formulation mathématique suivante du VRP est basé sur celle trouvée dans l'article de Bodin et al. (1983) [24] et Filipec et al. (1998) [25].

$$x_{ij}^k = \begin{cases} 1 & \text{Si le trajet de i à j est effectué par le vehicule k} \\ 0 & \text{sinon} \end{cases}$$

$$
(VRP)\begin{cases}
\text{Min } Z(X) = \sum_{i=0}^{N}\sum_{j=0,j\neq i}^{N}\sum_{k=1}^{K} c_{ij}x_{ij}^{k} \\
\sum_{i=0}^{N}\sum_{k=1}^{K} x_{ij}^{k} = 1 \quad \forall j \in \{1,\ldots,N\} & (1)\\
\sum_{j=0}^{N}\sum_{k=1}^{K} x_{ij}^{k} = 1 \quad \forall i \in \{1,\ldots,N\} & (2)\\
\sum_{i=0}^{N} x_{ip}^{k} - \sum_{j=0}^{N} x_{pj}^{k} = 0 \quad \forall p \in \{1,\ldots,N\}, k \in \{1,\ldots,K\} & (3)\\
\sum_{j=0}^{N} q_j(\sum_{i=0}^{N} x_{ij}^{k}) \leq Q_k \quad \forall k \in \{1,\ldots,K\} & (4)\\
\sum_{i=0}^{N}\sum_{j=0}^{N} d_{ij}x_{ij}^{k} \leq D_k \quad \forall k \in \{1,\ldots,K\} & (5)\\
\sum_{i=1}^{N} x_{0j}^{k} \leq 1 \quad \forall k \in \{1,\ldots,K\} & (6)\\
\sum_{i=1}^{N} x_{i0}^{k} \leq 1 \quad \forall k \in \{1,\ldots,K\} & (7)\\
x_{ij}^{k} \in \{0,1\} \quad \forall i,j,k.
\end{cases}
$$

La fonction objectif consiste à minimiser le coût total de transport. Les contraintes (1) et (2) assurent que chaque client soit servi par un seul véhicule. La continuité d'une tournée est assurée par la contrainte (3) où si un véhicule arrive à un point de livraison (client), il doit également le quitter. La contrainte (4) est la contrainte de capacité du véhicule. La contrainte (5) limite la distance maximale d'une tournée. Les contraintes (6), (7) assure que chaque véhicule ne soit pas utiliser plus d'une seule fois.

1.2 Complexité des problèmes Combinatoires

Les problèmes combinatoires sont intrigants car ils sont faciles à énoncer mais souvent très difficiles à résoudre. Par exemple, aucun algorithme n'existe pour trouver la solution optimale à un TSP en un temps polynomial. En effet pour le TSP (problème du voyageur de commerce), un voyageur a besoin de visiter beaucoup de villes. Il veut gagner du temps en adoptant la tournée la plus courte, plus on ajoute des villes plus le temps qu'il faut pour résoudre le problème de manière exact augmente de façon non polynomiale. Cela signifie que le temps pris pour résoudre ce problème augmente de façon exponentielle. Un solveur

exact pour ce problème, tel qu'un solveur qui effectue une enumération explicite (brute force search), prend un temps non polynomial. On dit que le problème TSP est NP-complet. Les termes "polynomial", "NP-complet" appartiennent au vocabulaire utilisés en Théorie de Complexité [26,27], cette théorie formalise l'idée intuitive de difficulté, elle permet de classer les problèmes en classes selon leurs niveau de difficulté.
Pour définir clairement la notion de complexité, il nous faut tout d'abord distinguer la notion de problème de la notion d'instance.

Définition 2 *Un Problème peut être définit comme :*

- *Un ensemble de données paramétrées ;*
- *Une requête spécifiant l'objectif de la résolution.*

Définition 3 *Une Instance d'un problème est obtenue en spécifiant des valeurs particulières à tous les paramètres du problème.Il ne faut donc pas confondre problème et instance d'un problème.*
Une instance d'un problème d'optimisation combinatoire est un couple $(S;f)$, où S est un ensemble fini de solutions possibles et $f : S \mapsto \mathscr{R}$ est une fonction qui affecte à chaque $s \in S$ une valeur $f(s)$. $f(s)$ est aussi appelée la fonction objectif.

Définition 4 *Un algorithme resout un problème si cet algorithme fournit une solution en un temps fini pour toute instance de ce problème. Ainsi, on peut définir la notion de decidabilité : Un problème est dit décidable s'il existe un algorithme capable de résoudre toutes ses instances en un temps fini*

Notons que le vocabulaire de la théorie de la complexité va à l'encontre du vocabulaire courant puisqu'une instance au sens de cette théorie est souvent appelée problème dans le langage courant.

Définition 5 *Une solution admissible ou réalisable est un élément de S*

Définition 6 *Une solution optimale (optimum global) s_{opt} est une solution admissible $s_{opt} \in S$ telle qu'il n'existe pas d'autre solution admissible s de S de coût inférieur (en minimisation) à celui de s_{opt}. s_{opt} est dite solution optimale de $(S;f)$. L'ensemble S_{opt} est l'ensemble de toutes les solutions optimales.*

Définition 7 *Le voisinage d'une solution est une fonction $V : S \mapsto 2^S$ qui affecte à chaque $s \in S$ un ensemble de voisins $V(s) \subseteq S$. $V(s)$ est appelé le voisinage de s.*

Définition 8 *Un optimum (minimum) local est une solution s^* telle que $\forall s \in V(s^*)$, $f(s^*) \leq f(s)$.*

Définition 9 *[28] Un problème d'optimization combinatoire admet trois versions qui sont :*

- ***La version recherche :*** *Etant donnée une instance $(S; f)$, trouver une solution optimale $s_{opt} \in S_{opt}$*
- ***La version evaluation :*** *Etant donnée une instance $(S; f)$, trouver la valeur optimale de la fonction objectif $f(s_{opt})$*
- ***La version decision :*** *Etant donnée une instance $(S; f)$ et une borne L, decider est ce qu'il existe ou pas une solution $s \in S$ avec $f(s) \leq L$*

Il est clair que la version recherche est la version la plus générale de ses trois version, puisque connaissant la solution optimale la version évaluation et la version décision seront trivial. S est l'ensemble de recherche, la finitude de S conduit à l'énumération comme approche intuitive de résolution, enumérer toutes les solutions et choisir celle qui réalise l'optimum. Cette approche n'est pas pratique pour les problèmes d'optimisation combinatoire pour cause de la taille très importante de l'ensemble S. Pour le problème de voyageur de commerce par exemple qui cherche le plus court chemin hamiltonien (qui passe par tous les sommets une seule fois) dans un graphe. L'algorithme aura donc à chercher tous les chemins possibles et de prendre le plus court. Or si on suppose qu'on a n sommets on a alors $\frac{(n-1)!}{2}$ chemins possibles, pour $n = 250$ au aura 10^{490} chemin possible. Imaginer le temps nécessaire pour les enumérer tous ! ! !

Les problèmes d'optimisation combinatoire sont des problèmes d'optimisation dont les ensembles réalisables sont finis mais combinatoires. Aussi, le nombre de solutions réalisables des problèmes combinatoires augmente exponentiellement en fonction de la taille du problème, et c'est ce qui exclut des méthodes de résolution basées sur l'énumération de toutes les solutions réalisables.

1.2.1 Niveaux de complexités

Trois niveaux de complexité peuvent être distinguer :

1. **Complexité d'un algorithme pour une instance**
   Ce niveau correspond au comportement d'un algorithme sur une instance particulière. Cette complexité est très simple à calculer, en effet elle correspond à la quantité

de ressources utilisées (nombre d'opérations élémentaires ou l'espace mémoire) pour résoudre une instance donnée par un certain algorithme.

2. **Complexité d'un algorithme**
   Le deuxième niveau généralise l'analyse de l'algorithme à toutes les instances du problème pour lequel il a été conçu. On parle alors de **cas moyen** ou de **pire cas** pour désigner le comportement de l'algorithme en moyenne (sur l'ensemble des instances) ou dans le cas de l'instance la plus défavorable (pour cet algorithme).
   On exprime la complexité d'un algorithme sous forme d'une fonction de taille des instances. Ainsi, la complexité de l'algorithme de tri rapide (Quick-Sort) est proportionnelle à n^2 dans le pire et $n\lg(n)$ en moyenne, n étant le nombre de nombres à trier. Cela signifie que cet algorithme nécessite , pour une liste de n éléments, un nombre d'opérations élémentaires maximum qui est de l'ordre de n^2.

3. **Complexité d'un problème**
   La complexité d'un problème se définit en utilisant la complexité, dans le pire cas, du meilleur algorithme capable de le résoudre. "Complexité dans le pire cas du meilleur algorithme" c'est cette notion qu'étudie principalement la théorie de la complexité.

Définition 10 ***Algorithme polynomial***
On dit qu'un algorithme est polynomial lorsque sa complexité dans le pire cas est bornée supérieurement par un polynôme en la taille des instances traitées. Par exemple, le tri rapide (Quick sort) est un algorithme polynomial puisque sa complexité dans le pire cas est en n^2. Le tri par tas est également polynomial puisque sa complexité dans le pire cas est en $n\lg(n)$, fonction qui peut être bornée supérieurement par n^2.

Définition 11 ***Problème polynomial***
Un problème polynomial est un problème qui se résout par un algorithme polynomial.

1.2.2 Classes de Complexité

La théorie de la complexité s'interesse principalement aux problème de décision c'est à dire les problème pour lesquels la réponse est soit OUI soit NON. Elle ne traite pas directement les problèmes d'optimisation, neanmoins on peut ramener un problème d'optimisation à une suite de problème de décision. Le problème consistant de trouver la valeur optimale peut ainsi se ramener à une série de problème de décision du type "existe il une solution de coût inférieur à α ?" (dans le cas d'une minimisation).

Le but de la théorie de la complexité est la classification des problèmes de décision suivant leur degré de difficulté de résolution. Dans la littérature, il existe plusieurs classes de complexité, mais les plus connues sont les suivantes :

- **La classe P** : C'est la classe des problème pouvant être résolus en un temps polynomial. Les problèmes appartenant à cette classe sont dits faciles.

- **La classe NP** : C'est l'abréviation pour "non-deterministic polynomial time". Un problème appartient à la classe NP si l'on peut toujours vérifier en temps polynomial qu'une solution potentielle vérifie l'ensemble des propriétés que doit satisfaire une solution du problème. On parle dans ce cas de validation en temps polynomial. On a clairement, P $\subset$ NP. L'inclusion dans l'autre sens est un problème ouvert (qui peut rapporter des millions). La communauté des chercheurs semblent penser que P $\subsetneq$ NP mais aucune preuve n'existe.

  Au sein de la classe NP on distingue l'ensemble des problèmes les plus difficiles de NP c'est la classe des problème NP-Complet. Exemple de problème dans NP : le problème SAT (de satisfiabilité). Soit $F = (x_1, x_2, \ldots, x_n)$ une expression logique de n variables. Le problème SAT consiste à trouver une valeur vrai ou faux pour chacune des variables x_i de telle manière à rendre vrai l'expression $F = (x_1, x_2, \ldots, x_n)$.

Définition 12 ***Reduction polynomiale***
Soient $Q^{'}$ ET Q deux problèmes de décision, on dit que $Q^{'}$ se réduit polynômialement en Q s'il existe un algorithme A tel que si $I^{'}$ est une instance de $Q^{'}$, l'algorithme transforme cette instance en temps polynômiale en une instance I du problème Q, ces instances étant telles que $I^{'} \in Q^{'} \Longleftrightarrow I \in Q$

Définition 13 ***NP-Complet***
Un problème de décision est dit NP-complet s'il est dans NP et si tout problème de la classe NP lui est polynomialement réductible. Les problème dit NP-complet sont des problèmes NP qui ont une forme de propriété universel : si on sait résoudre un problème NP-complet, alors on sait tous les résoudre avec esseciellement la même complexité.

Donc si on sait résoudre un poblème NP-complet en temps polynômiale, on sait résoudre tous les problèmes de la classe NP en temps polynômiale et P = NP. Par contre si on montre qu'un poblème NP-complet n'admet pas d'algorithme de résolution en temps polynômiale, alors P $\neq$ NP et qu'aucun problème NP-complet n'admet d'algorithme de résolution en temps polnômiale. Une liste d'une centaine de problèmes NP-complet est présentée dans le livre de M.R.Garey and D.S. Johnson (1979) "Computer and intractability" [29].

Théorème 1 ***Théorème de Cook 1971*** *[30]*
Le problème de satisfiabilité est NP-complet .

Cook en 1971 à démontré dans l'article intitulé " The Complexity of Theorem Proving Procedures" [30] que tous les problèmes de la classe NP sont réductibles au problème de la satisfiabilité d'une expression logique quelconque. Autrement dit, trouver un algorithme polynomial pour le problème de SAT, c'est avoir une réponse à la question : la classe P est elle égale à la classe NP ?

Définition 14 ***problème NP-difficile***
Un problème de décision est NP-difficile ssi il est au moins aussi difficile que n'importe quel problème de décision dans NP. Autrement dit, un problème de décision est NP-difficile s'il est situé à l'extrémité de NP ou au-delà (voir la figure1.1). Il convient de noter qu'un problème de décision NP-difficile ne se trouve pas forcément dans NP, mais qu'il peut se retrouver dans une classe de complexité supérieure.

Remarque 1 *On peut dire alors qu'un problème de décision est NP-complet ssi 1) il appartient à la classe de problèmes de décision NP et 2) il est NP-difficile.*

Pour les problèmes NP-difficiles, deux possibilités pour leur résolution sont offertes. Si le problème est de petite taille, alors un algorithme exact permettant de trouver la solution optimale peut être utilisé (procédure de séparation et évaluation (Branch and Bound), programmation dynamique ...). Malheureusement, ces algorithmes par nature énumératifs, souffrent de l'explosion combinatoire et ne peuvent s'appliquer à des problèmes de grandes tailles. Dans ce cas, il est nécessaire de faire appel à des heuristiques ou métaheuristiques permettant de trouver de bonnes solutions approchées.

1.3 Methodes de résolution des problèmes d'Optimisation Combinatoire

Les méthodes de résolution des problèmes d'optimisation combinatoire se divise en deux type exactes et approchés, cette classification est faite selon la qualité de la solution trouvée par la méthode de résolution, si la solution est optimale exacte on parle de méthode exacte, si par contre la solution est approchée on parle de méthode heuristique ou métaheuristique.

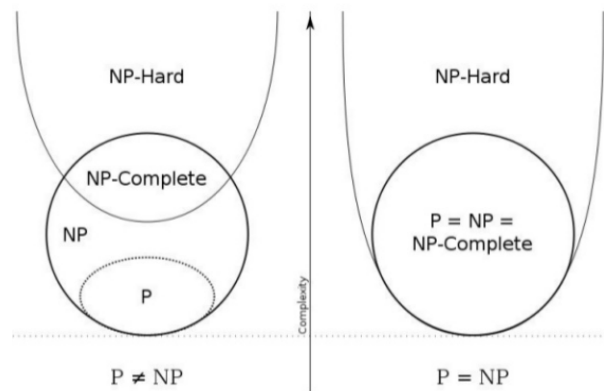


FIGURE 1.1 – Diagramme d'Euler pour représenter les classes de problèmes P, NP, NP-complete, and NP-hard.

1.3.1 Methodes de résolution exactes

Pour les problèmes ayant un ensemble fini de solutions admissibles, un simple algorithme exact consiste simplement à énumerer toutes les solutions de l'ensemble des solutions possibles. Un tel algorithme n'est pas pratique dans le cas des problèmes d'optimisation combinatoire en raison de la grande taille de l'ensemble des solutions admissibles. Pour augmenter leur éfficacité, toutes les méthodes exactes modernes utilisent des règles d'élimination de parties de l'espace de recherche dans lesquelles la solution (optimale) ne peut pas exister. Ces approches font une énumération implicite de l'espace de recherche. Les méthodes exactes les plus connues sont Branch and Bound, Branch and cut, Branch and Bound and Cut, Programmation Dynamique dévelopée par Richard Bellman [31].

Les méthodes (ou algorithmes) exactes garantissent de trouver la solution optimale d'un problème d'optimisation combinatoire donné mais le temps nécessaire pour son obtention augmente de manière exponentielle en augmentant la taille du problème. De façon pratique, seuls les problèmes de petites et moyennes tailles peuvent être résolus de façon optimale par des algorithmes exacts. De plus, pour certains problèmes, la consommation de mémoire de ces algorithmes peut être très grande et peut parfois entraîner l'arrêt prématuré de l'application informatique.

Pour plus de details voir [32] un survey des méthodes exactes appliquées au problèmes NP-complet. Dans ce qui suit nous allons citer quelques unes d'elles.

1.3.1.1 Branch and Bound

La téchnique de branch and Bound est une technique de simulation des solutions possibles d'un problème donné sans avoir à les enumérer toutes une à une, Depuis sa création par Land and Doig en 1960 [33], elle a été testée sur plusieurs problèmes NP-difficiles de l'optimisation combinatoire. Le premier testé est le problème de voyageur de commerce (TSP) par Little et al. en 1963 [34]. La méthode Branch and bound partitionne le problème global en sous- problèmes, elle réduit l'espace de recherche par l'utilisation du principe "diviser pour regner". Deux choses sont nécessaires dans l'algorithme de séparation et d'évaluation :

- **Processus de séparation (Branching process) :** Un ensemble de solutions, représenté par un noeud, peut être partitionné en ensembles mutuellement exclusifs. Chaque sous-ensemble de la partition est représenté par un enfant du noeud d'origine.
- **Processus de calcul de borne :** Calcul de limite inférieure (Lower bounding), un algorithme est utilisé pour calculer une borne inférieure du coût de n'importe quelle solution dans un sous-ensemble donné.

Pour plus de detail sur la méthode voir le livre de Christos H. Papadimitriou, Kenneth Steiglitz Combinatorial Optimization Algorithms and Complexity [35].

L'algorithme 1 représente l'algorithme de branch and bound basique tel quel'il a été présenté dans le livre de Christos H et al. [35].
l'ensemble $activeset$ est utilisé pour sauvegarder les noeuds vivants. La variable U est utilisée pour sauvegarder le coût de la meilleure solution complete (U est la borne supérieure de la solution optimale).

Algorithm 1 L'algorithme de branch-and-bound

Begin
$activeset := 0$; (comment : "0" est le problème original).
$U := \infty$;
$currentbest := anything$;
while $activeset$ est non vide **do**.
Begin
Choisir un noeud de séparation (branching) $k \in activeset$;
Retirer le noeud k de la liste $activeset$.
Générer les enfants du noeud k, enfant i, $i = \{1, \ldots, n_k\}$
Générer les bornes inférieures correspondantes z_i ;
For $i = \{1, \ldots, n_k\}$
Begin
if $z_i \geq U$ **then** kill $child$ i
else if enfant i est une solution complete **then**
$U := z_i$,
$currentbest := child$ i
Else ajouter $child$ i à $activeset$
end if
End
end while
End

1.3.1.2 Branch and Cut

Les algorithmes exacts de branch and cut peuvent être utilisés dans de nombreux problèmes d'optimisations combinatoires et ils peuvent fournir une garantie d'optimalité. Les méthodes de branch and cut sont des algorithmes exacts, qui résultent de la combinaison des méthodes de coupe proposé par Gomory (1963) [36] et de l'algorithme de branch and bound [37]. Cette méthode a montré une efficacité meilleure par rapport à la méthode de branch and bound face à des instances de grande taille d'un problème donnée [38, 39].

1.3.1.3 Branch and Price

La méthode de Branch and Price [40, 41] est considérée aussi comme une généralisation de la méthode Branch and Bound. Plus précisément dans un branch and price on a une exploration arborescente de la même façon que dans le branch and bound mais au lieu d'avoir une simple résolution de programme linéaire à chaque nIJud on a à résoudre un programme linéaire avec génération de colonnes. Voir le survey de Desrosiers et al [42] pour plus de détails.

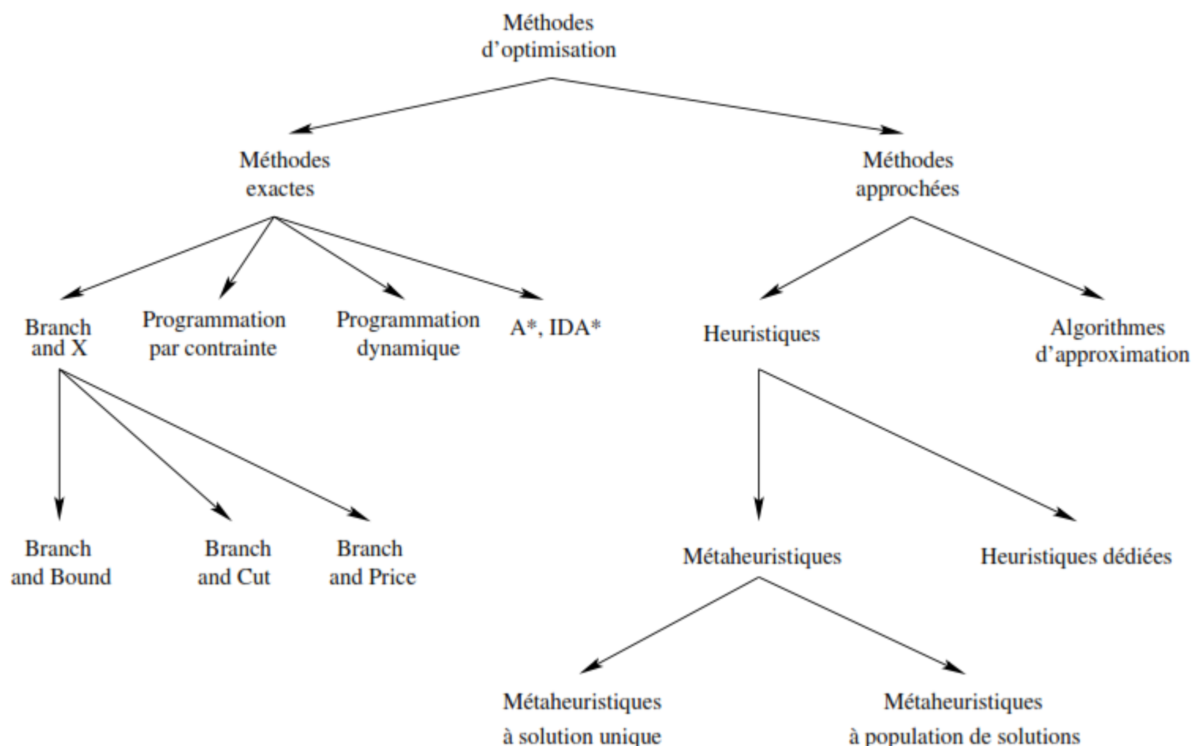


FIGURE 1.2 – Une possible hièrarchie des méthodes d'optimisation [9]

1.3.2 Methodes de résolution approchées

Nicholson [41] dit qu'une méthode heuristique peut être définie comme étant une procédure exploitant au mieux la structure du problème considéré dans le but de trouver une solution de qualité raisonnable dans des délais aussi courts que possible. D'autres méthodes existent, appelées métaheuristiques, qui représentent des frameworks ou cadres généraux de résolution approchée et peuvent être appliquées à plusieurs catégories de Problèmes d'Optimisation Combinatoire.
Un bon aperçu de ces méthodes est donné par Widmer [43] et une bibliographie répertoriant une grande quantité de travaux a été réalisée par Osman et Laporte [42].
Dans le chapitre qui suit nous présentons quelques méthodes métaheuristiques et hybrides destinées à la résolution des problèmes d'optimisation combinatoire.

1.3.2.1 Les heuristiques constructives [1]

Les heuristiques constructives sont généralement considérées comme les méthodes approchées les plus rapides. La solution dans le cas d'utilisation d'une heuristique constructive est générée (ou construite) en ajoutant à chaque étape, une composante à une solution partielle

(initialement vide). On continue jusqu'à ce qu'on obtient une solution complete ou jusqu'à ce qu'un autre critère d'arrêt soit satisfait.
Les heuristiques constructives sont caractérisées par un mécanisme de construction qui spécifie pour chaque solution partielle s_p l'ensemble des extensions possibles. Cet ensemble est noté $Ext(s_p)$, est un sous ensemble de l'ensemble complet des composantes d'une solutions noté C. À chaque étape de construction, un élément de $Ext(s_p)$ est séléctionné et est ajouté à s_p, ceci est fait jusqu'à avoir $Ext(s_p) = \emptyset$, cela revient à un des deux cas : Soit on est arrivé à avoir une solution complete ou bien que s_p est une solution partielle qui ne peut pas être étendue à une solution complete. Le framework algorithmique d'une heuristique constructive est montré par l'algorithme 2.

Algorithm 2 Heuristique constructive [1]

$s^p = \emptyset$
while $Ext(s^p) \neq \emptyset$ **do**
  $c \leftarrow Selectionner(Ext(s^p))$
  ajouter c à s^p
end while

L'exemple le plus connu d'une heuristique constructive est l'heuristique gloutonne, qui utilise une fonction dite gloutonne afin de choisir une composante d'une solution de l'ensemble $Ext(s^p)$ à chaque étape. Une fonction gloutonne est une mesure de la qualité des composantes de la solution, c'est-à-dire un indicateur de l'impact sur la valeur de l'objective lorsqu'une composante serait inclus.

1.3.2.2 La recherche locale [1]

La recherche locale (LS) commence par par une solution initiale et à chaque étape, on remplace la solution courante par une solution meilleure à partir d'un voisinage bien défini de cette solution courante. Ce processus s'arrête dès que le voisinage de la solution courante ne contient plus une solution qui est meilleure que cette dernière.
Le voisinage d'une solution x est définit par une structure de voisinage. Cette structure de voisinage est, dans de nombreux cas, définie implicitement en spécifiant les modifications qui doivent être appliquées à une solution x afin de générer tous ses voisins. L'application d'un tel opérateur qui produit un voisin $x^{'} \in V(x)$ d'une solution x est connu sous le nom de mouvement.
Une version simpliste de la méthode de recherche local est présenté dans l'algorithme 3.

Algorithm 3 Recherche locale [1]

Données Solution initiale x, Une structure de voisinage V.
while $\exists x^{'} \in V(x)$ telle que $f(x^{'}) < f(x)$ **do**
  $x \leftarrow ChoisirUnVoisinMeilleur(V(x))$
end while

Il existe deux techniques principales possibles de définition de la fonction de choix de la solution voisine qui apporte une amélioration par rapport à la solution courante (fonction $ChoisirUnVoisinMeilleur(V(x))$ dans l'algorithme (3). La première technique est appelée "première amélioration", en utilisant cette technique le voisinage de la solution courante est examiné dans un ordre prédéfini et la première solution dans ce voisinage qui est meilleure que x est retournée. Par contre, la deuxième technique "la technique meilleure amélioration" revient à faire une exploration exhaustive du voisinage et de retourner la meilleure solution trouvée dans $V(x)$. Noté que les deux techniques de l'algorithme de recherche local s'arrêre dans un minimum local. En général, la performance d'une méthode de recherche locale dépend fortement de la définition de la structure de voisinage V.

1.3.2.3 Les métaheuristiques

Le terme métaheuristique a été introduit pour définir des méthodes heuristiques qui peuvent être adapter à la résolution de n'importe quel problème. En d'autres termes, une métaheuristique peut être vu comme un framework général algorithmique qui peut être appliqué à différents problèmes d'optimisation avec relativement de petites modifications pour la rendre adaptée à un problème spécifique [44]. Les métaheuristique sont des algorithmes approchées qui essentiellement essaye de combiner des heuristiques constructives et/ou des méthodes de recherche locales avec d'autres idées et les mettre dans un framework de haut-niveau dans le but d'explorer au mieux l'espace de recherche pour trouver une bonne solution au problème traité [1, 45, 46].

Dans ce chapitre nous avons donné une vue globale sur les problèmes d'optimisation combinatoire, leurs modélisation, complexité et méthodes de résolution exactes et approchées. Dans le chapitre suivant nous nous orientons vers les méthodes approchées et nous donnerons plus de détails sur ses dérnières.

Dans la vie, rien n'est à craindre, tout est à comprendre.

Marie Curie

CHAPITRE 2 LES MÉTAHEURISTIQUES ET MÉTHODES HYBRIDES

Le terme Métaheuristique a été introduit pour la première fois par Glover [47], composé de deux mots grecs : Heuristique qui derive du verb heuriskein signifiant "trouver", et le suffix meta qui veut dire, "au-delà, dans un niveau supérieur", Avant que ce terme ne soit largement adopté, les métaheuristiques étaient souvent appelées heuristiques modernes [48].

Les Métaheuristiques sont des méthodes d'optimisation qui peuvent être adaptées à la résolution de manière approchée de n'importe quel problème d'optimisation. Ce sont des algorithmes de nature heuristique comme leur nom indique ce qui les differencie des méthodes exactes, qui ; non seulement trouvent des solutions exactes mais aussi garantient que la solution optimale d'un problème donné sera trouvée dans un temps fini (bien que souvent prohibitif). Les métaheuristiques sont donc développées dans le but de trouver une solution "assez bonne" dans un temps de calcul "assez petit". En conséquence, ils ne sont pas soumis à une explosion combinatoire (le phénomène où le temps de calcul nécessaire pour trouver la solution optimale des problèmes NP-difficiles augmentent de manière exponentielle en fonction de la taille du problème).

Jusqu'à présent, il n'y a pas clairement de consensus sur la définition exacte des métaheuristiques. Dans ce qui suit nous présentons quelques propositions de définition de certains chercheurs.

D'après Glover [47] "Une métaheuristique est un processus itératif qui guide et modifie les opérations des heuristiques subordonnées afin de produire efficacement des solutions

de haute qualité. Il peut manipuler une solution unique complète (ou incomplète) ou un ensemble de solutions à chaque itération. Les méthodes heuristiques subordonnées peuvent être des procédures de niveau haut (ou bas), une simple recherche locale ou simplement une méthode de construction."

Selon Osman et Laporte [49] "Une métaheuristique est définie comme un processus itératif qui guide et modifie les opérations des heuristiques subordonnées pour produire efficacement des solutions de haute qualité. Elle peut combiner intelligemment différents concepts pour explorer l'espace de recherche et utiliser des stratégies d'apprentissage pour structurer l'information.

Sörensen et Glover ont proposés en 2013 [50] une autre définition : "Une métaheuristique est une plateforme algorithmique de haut niveau indépendante du problème, qui fournit un ensemble de directives ou de stratégies pour développer des algorithmes d'optimisation heuristique. Le terme méta heuristique peut être utilisé pour désigner une implémentation spécifique d'un algorithme heuristique d'optimisation pour la résolution d'un problème donné selon des directives exprimées dans une telle plateforme."

Les métaheuristiques sont des procédures générales de haut niveau qui combinent des heuristiques simples et des règles pour trouver des solutions de bonne qualité aux problèmes d'optimisation combinatoire.
Donc pour résumer Le terme "Métaheuristique" a été utilisé par la communauté scientifique pour exprimer deux choses differentes :

- Certains ont utilisé le terme Métaheuristique pour désigner une plateforme de haut niveau (high-level framework), qui représente un ensemble de concepts et de stratégies qui fonctionnent ensemble pour offrir de nouveaux algorithmes d'optimisation.
- D'autres chercheurs utilisent le terme "Métaheuristique" dans le sens d'une implementation spécifique d'un algorithme basé sur une plateforme déjà définie (ou une combinaison de concepts pris de différentes plateformes) conçu pour trouver une solution à un problème d'optimisation spécifique.

2.1 Classifications des Métaheuristiques

Plusieurs manières de classifier les métaheuristiques existent, une classification dépend des charactéristiques selectionnées pour diffirencier les métaheuristiques entre elles. Quelques classifications qui existent dans la literature sont présentées briévement dans ce qui suit :

1. **Classification basée sur les origines d'une métaheuristique :** C'est la classification la plus intuitive, basée sur les origines de la métaheuristique en question, on a les métaheuristiques inspirées de la nature comme les algorithmes génétiques (GA : Genetic Algorithms) Holland en 1975 [51], et l'algorithme de la colonie de fourmie (ACO : Ant Colony Optimization) Dorigo et al., en 1991 [52–54], et d'autres qui ne sont pas inspirés de la nature telles que La recherche tabou (TS :Tabu Search) Glover en 1986 [47] et La recherche locale itérée (ILS : Iterated local search) Stützle en 1998 [55].

2. **Classification basée sur le nombre de solutions manipulées par une métaheuristique :** On distingue deux classes Les métaheuristiques à solution unique et Les métaheuristiques à population de solutions. Les métaheuristiques à solution unique appelées aussi méthodes de trajectoire. Contrairement aux métaheuristiques à base de population qui améliorent une population de solutions tout au long du processus de recherche telles que les algorithmes évolutionnaires, les métaheuristiques à solution unique commencent avec une seule solution initiale et procede à son amélioration progressivement, en construisant une trajectoire dans l'espace de recherche. Les méthodes de trajectoire englobent essentiellement recherche tabou, le recuit simulé [56], la méthode de descente, la méthode GRASP [57, 58], la recherche à voisinage variable [8, 59], la recherche locale itérée [55].

3. **Classification basée sur la nature de la fonction à optimiser :** La fonction à optimiser ou fonction objectif d'un problème donné peut être statique ou dynamique. Les métaheuristiques peuvent également être classées en fonction de la manière dont elles utilisent la fonction objectif. Alors que certains algorithmes gardent la fonction objectif du problème sans changement ou modification, d'autre métaheuristiques comme la recherche locale guidée (Guided Local Search) qui modifie la fonction objectif tout au long du processus de recherche dans le but de s'échapper des optima locaux.

4. **Classification selon le nombre de structure de voisinage utilisées :** La plupart des metaheuristiques utilisent une seule structure de voisinage, d'autres comme la recherche à voisinage variable (VNS : Variable neighborhood search) Mladenovic en 1995 [59], Mladenovic et Hansen en 1997 [8].

5. **Classification sur base d'utilisation ou non d'une mémoire :** Une caractéristique très importante pour classer les métaheuristiques est l'utilisation qu'ils font de l'historique de recherche, c'est-à-dire s'ils utilisent une mémoire ou pas et selon ce critère on peut classer les métaheuristiques en deux classes métaheuristique à mémoire et métaheuristiques sans mémoire. Les métaheuristiques qui font usage de l'historique de la recherche peuvent le faire de diverses manières. On différentie généralement les méthodes ayant une mémoire à court terme de celles qui ont une mémoire à long terme.

6. **Classification sur base de présence ou non d'aspect probabilistique :** Les métaheuristiques peuvent ausi être classées en deux classes : Métaheuristiques Deterministes versus métaheuristiques stochastique. Les méthodes stochastiques permettent des sauts probabilistes depuis les solutions courantes vers les prochaines solutions. Dans les méthodes stochastiques, un même point de départ peut conduire à des solutions différentes puisque des paramètres aléatoires sont utilisés au cours du processus. Dans les métaheuristiques déterministes, au contraire, une solution initiale spécifique conduit toujours à la même solution finale. En effet, Une méthode déterministe parcourt l'espace de recherche toujours de la même manière, tandis que deux exécutions d'une méthode stochastique peuvent fournir des résultats différents.

Dans ce qui suit nous présentons certaines métaheuristiques selon le nombre de solutions manipulées, métaheuristiques à solution unique et métaheuristiques à population de solution.

2.2 Métaheuristiques basées sur des métaphores

La majorité des métaheuristiques qui existent dans la litératture ont été inspirées de métaphores. Dans le tableau qui suit nous citons quelques unes d'entre elles apparues durant la période de 1954 à 2018.
Selon l'etude statistique faite par Kashif et al. [10] sur le nombre et la nature des métaheuristique qui existent dans la littérature. Les métaphores des métaheuristiques disponible aujourd'hui dérivent de plusieurs disciplines notamment la biologie, physique, Chimie, La vie quotidienne, etc. La figure2.1 montre le pourcentage d'utilisation d'une discpline donnée comme métaphore pour création de nouvelles métaheuristiques.

TABLE 2.1 – Quelques Métaheuristiques de 1954 à 2018.

Num	Métaheuristique	Année	Auteurs
1	Evolution Process(EP)	1954	Barricelli [60]
2	Random Search(RS)	1963	Rastrigin [61]
3	Random Optimization(RO)	1965	Matyas [62]
4	Evolutionary Programming	1966	Fogel et al. [63]
5	Graph Partioning Method(GPM)	1970	Kernighan and Lin [64]
6	Genetic Algorithm(GA)	1975	Holland [65]
7	Scatter Search(SS)	1977	Glover [66]
8	Metaplan	1978	Mercer and Sampson [67]
9	Simulated Annealing(SA)	1983	Kirkpatrick et al. [56]
10	Tabu Search(TS)	1986	Glover [47]
11	Immune System Algorithm (IA)	1986	Farmer et al. [68]
12	Memetic Algorithm(MA)	1989	Moscato [69]
13	Ant Colonny Optimization(ACO)	1992	Dorigo [52]
14	Multi-Objective GA(MOGA)	1993	Fonseca and Fleming [70]
15	Reactive Search Optimization(RSO)	1994	Battiti and Brunato [71], [72]
16	Particle Swarm Optimization(PSO)	1995	Kennedy and Eberhart [73]
17	Differential Evolution(DE)	1997	Storn and Price [74]
18	Harmony Search(HS)	2001	Geem et al. [75]
19	Bees Optimization	2004	Nakrani et Tovey [76]
20	Artificial Bee Colony Algorithm(ABC)	2005	Karaboga [77]
21	Glowworm Swarm Optimization(GSO)	2005	Krishnanand and Ghose [78]
22	Honey bee Mating Optimization(HbMO)	2006	Haddad et al. [79]
23	Imperialist Competitive Algorithm(ICA)	2007	Atashpaz-Gargari [80]
24	Intelligent Water Drops(IWD)	2007	Shah-Hosseini [81]
25	Firefly Algorithm(FA)	2008	Yang [82]
26	Cuckoo Search(CS)	2009	Yang and De [83]
27	Gravitational Search Algorithm(GSA)	2009	Rashedi et al. [84]
28	League Championship Algorithm(LCD)	2009	Kashan [85]
29	Bat Algorithm (BA)	2010	Yang [86]
30	Galaxy Based Search Algorithm(GbSA)	2011	Shah-Hosseini [87]
31	Spiral Optimization(SO)	2011	Tamura and Yasuda [88] [89]
32	Teaching Learning Based Optimization(TLBO)	2011	Rao et al. [90]
33	Krill Herd(KH)	2012	Gandomi and Alavi [91]
34	Coral Reefs Optimization(CRO)	2013	Salcedo-Sanz et al. [92]
35	Swallow Swarm Optimization(SSO)	2013	Neshat et al. [93]
36	Interior Search Algorithm(ISA)	2014	Gandom [94]
37	Gradient Evolution Algorithm(GEA)	2015	Kuo and Zulvia [95]
38	Lion Optimization Algorithm(LOA)	2015	Yazdani and Jolai [96]
39	Optics Inspired Optimization(OIO)	2015	Kashan [97]
40	Water Wave Optimization(WWO)	2015	Zheng [98]

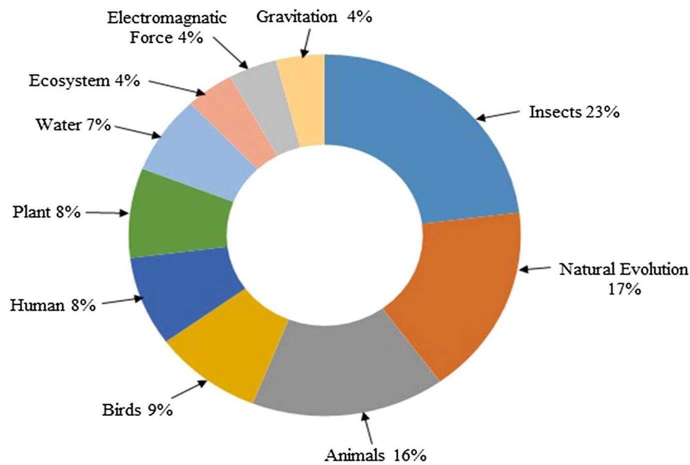


FIGURE 2.1 – Métaphore utilisées par les chercheurs pour le dévelopement de nouvelles métaheuristiques. [10]

TABLE 2.1 – (*Suite Tableau2.1*).

41	Duelist Algorithm	2016	Biyanto et al. [99]
42	Killer Whale Algorithm	2016	Biyanto et al. [100]
43	Rain Water Algorithm	2017	Biyanto [101]
44	Hydrological Cycle Algorithm	2017	Wedyan et al. [102]
45	Monarchy Metaheuristic	2019	Ahmia and Aider [13]

2.3 Concepts importants des métaheuristiques

Dans ce qui suit nous présentons les plus importants concepts à prendre en consédération lors de la création d'une métaheuristique.

1. Une métaheuristique doit commencer par définir sa fonction objective et une manière de représenter ses solutions c'est à dire comment la solution est codée dans l'algorithme et comment la qualité d'une solution est mesurée.

2. Une métaheuristique doit garder un equilibre entre les deux principes principals qui sont l'intensification et la diversification.

    - **La diversification** est un processus d'exploration de l'espace de recherche, il permet de découvrir de nouvelles régions de l'espace de recherche ce qui permet d'éviter d'être bloquer sur des optima locaux mais ce processus ne favorise pas la convergence de la métaheuristique.

- **L’intensification** est l’exploitation de l’information accumulée durant la recherche et concentration sur une zone précise où on a plus de chance de croiser l’optimum global. Intensification signifie améliorer les meilleures solutions ou explorer leurs voisinages dans l’espoir de trouver des solutions meilleures.

  La diversification donne à la métaheuristique un comportement de recherche globale. Tandis que l’intensification donne à la métaheuristique une caractéristique de recherche locale. L’intensification fait référence à la capacité des opérateurs de la métaheuristique à utiliser les informations disponibles à partir des solutions des autres itérations et d’intensifier la recherche dans de bonne zone pour favoriser la convergence de la métaheuristique, le risque est alors de provoquer une convergence prématurée, par exemple en attirant toutes les solutions vers un optimum local. Pour une métaheuristique, la difficulté est de réaliser un équilibre entre ces deux principes afin de converger vers l’optimum global de l’espace de recherche, tout en évitant de rester bloqué sur un optimum local.

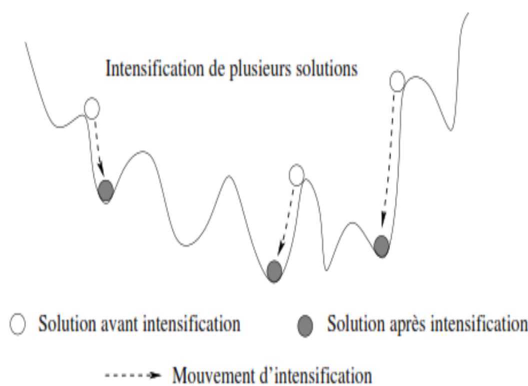


FIGURE 2.2 – Intensification [9]

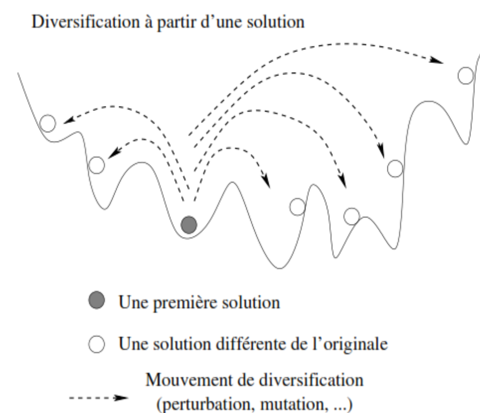


FIGURE 2.3 – Diversification [9]

Il est important de préciser que les termes exploration et exploitation sont parfois utilisés à la place de diversification et intensification, l’ensemble des spécialistes des métaheuristiques accepte et utilise le plus souvent les termes diversification et intensification.

L’équilibre entre diversification et intensification tel que mentionné ci-dessus est important, d’une part pour accélerer la detection des régions de recherches qui contiennent de bonnes solutions et d’une autre part pour ne pas perdre trop de temps dans

des régions de l'espace de recherche qui sont déjà explorées ou qui ne fournissent pas de bonnes solutions.

3. Lors de la mise en oeuvre d'une métaheuristique, la définition de la structure de voisinage utilisée est une tache pertinente. Le concept de voisinage définit des solutions qui sont, dans une certaine mesure, prochent les unes des autres.

4. Souvent la performance d'une métaheuristique s'influence par le changement de la solution de démarrage. Les solutions initiales peuvent être générées aléatoirement ou de manière heuristique.

Algorithm 4 Cadre générique d'une Métaheuristique [7]

Elements nécessaires (Stratégie de séléction)SS, (Stratégie de génération) SG, (Stratégie de remplacement) SR, (Stratégie de mise à jour) SM, (Condition d'arrêt) *CA*.
Créer un ensemble S de n solutions.
Evaluer chaque solution de l'ensemble S.
Identifier x_{best} la meilleure solution de S.
while *CA* n'est pas encore vérifié **do**
  Séléctionner un nombre μ de solutions de S suivant SS.
  Générer un nombre λ de solutions nouvelles à partir des μ solutions séléctionnées.
  Choisir un nombre θ de solutions de l'ensemble S et utiliser SR pour remplacement.
  Mettre à jour l'ensemble S en utilisant la SM.
  Evaluer les solutions de l'ensemble S.
  Identifier la nouvelle meilleure solution de S.
  Mettre à jour x_{best}.
end while
Retourner x_{best}

La majorité des métaheuristiques suivent une séquence similaire d'opérations, et c'est pourquoi elles peuvent être définies par un cadre (framework) générique commun présentée par l'Algorithme 4 [7].

2.4 Les métaheuristiques à solution unique

Les métaheuristiques à solution unique (les S-métaheuristiques), appelées aussi les métaheuristiques de trajectoire et cette appellation est due à la trajectoire formée au cours du processus de recherche par l'évolution de la solution de démarrage. En effet, ces méthodes démmarent avec une solution unique et la remplace à chaque itération par une autre solution suivant un certain mecanisme, le processus de recherche s'arrête dès qu'un critère

d'arrêt est atteint.
Dans ce qui suit nous présentons un ensemble de méthodes de trajectoire en commençant par les plus connus notamment : La recherche tabou, le recuit simulé, la méthode de descente, la méthode GRASP, la recherche à voisinage variable.

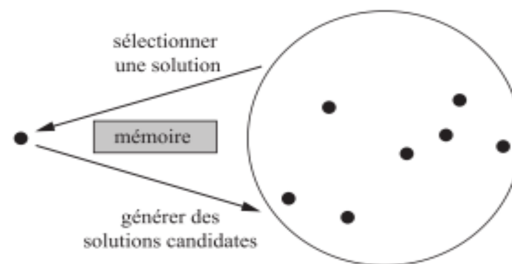


FIGURE 2.4 – Illustration du fonctionnement d'une métaheuristique à base de solution unique (S-métaheuristique) [11].

2.4.1 La méthode de descente (Hill Climbing)

La méthode de descente (appelée aussi "Hill Climbing" pour les problèmes de maximisation) est une méthode ancienne et très répandu et ceci est du à sa simplicité et sa rapidité.
Elle est basée sur la définition de structure de voisinage et donc à chaque fois qu'on change cette structure on developpe une nouvelle variante de cette méthode. La méthode commence par une solution initiale générée aléatoirement ou par une méthode heuristique. *À* chaque pas de la recherche, cette méthode progresse vers une solution voisine de meilleure qualité, elle s'arrête quand tous les voisins candidats sont moins bons que la solution courante, c'est-à-dire lorsqu'un optimum local est atteint.

La selection de la solution voisine peut se faire par plusieur façon : la solution voisine retenue peut être la meilleure solution de l'ensemble du voisinage, ou bien juste la première solution croisée dans l'ensemble de voisinage meilleure que la solution courante. Le pseudo code de la méthode de descente est présenté par l'Algorithm 5.

Algorithm 5 **Pseudo code de la méthode de descente**

Générer une solution initiale x
Générer un ensemble $V(x)$ de voisins de x
Initialiser la meilleure solution x^* par x
for Chaque voisin $x^{'}$ dans $V(x)$ **do**
  Evaluer $x^{'}$
  if $x^{'}$ est meilleure que x **then**
    Mettre à jour x^* par $x^{'}$
  end if
end for
Retourner x^*

2.4.2 La Recherche Tabou

Fred Glover en 1986 [47] a proposé une méthode déterministe appelé recherche tabou, la caractéristique principale de cette méthode est l'utilisation d'une mémoire. Basée sur le principe de la mémoire humaine, cette méthode a rendu possible la mémorisation des solution croisées précedemment au cours du processus de recherche en les stockant dans ce qui est appelé la liste tabou. L'utilisation de cette liste tabou a pour but d'éviter les retours en arrière (mouvements cycliques), elle contient des mouvements ou des solutions qui sont temporairement interdits. La taille de cette liste tabou représente le paramètre le plus important de cette métaheuristique. En effet, cette liste permet à l'algorithme d'éviter de retomber sur des solutions déjà croisées et donc éviter le processus de cyclage en forçant l'algorithme d'accepter dans certaines itérations des solutions moins bonnes que la solution courante.

Algorithm 6 **Pseudo code de l'Algorithme de Recherche Tabou**

Génerer une solution initiale s
Fixer la taille de la liste tabou T
while la condition d'arrêt n'est pas vérifiée **do**
  Générer N non-tabu voisins de s
  Evaluer les N voisins
  Selectionner le meilleur voisin $s^{'}$
  Mettre à jour la meilleure solution déjà trouvée s^*
  Inserer $s^{'}$ dans la liste T.
  Remplacer s par $s^{'}$
end while
Retourner s^*

La taille de la liste tabou contrôle la mémoire du processus de recherche. Avec une taille trop petite la recherche sera concentré sur des régions trop petite et donc le mécanisme d'exploitation (l'intensification) est favorisé, dans le cas contraire la liste tabou force l'algorithme

à explorer des régions plus larges et donc le mécanisme d'exploration (diversification) est favorisé.

2.4.3 Le Recuit Simulé

Présentée par Kirkpatrick et al. en 1983 [56], c'est l'une des plus anciennes métaheuristique et la première qui a décrit une stratégie qui permet de s'échapper des optima locaux. Les origines de cette métaheuristique reviennent à l'algorithme Metropolis proposé en 1953 [103]. Elle s'inspire d'un processus utilisé en métallurgie. Le recuit en métallurgie consiste à appliquer des cycles de chauffage et de refroidissement contrôlés à un matériau, afin de réorganiser sa structure cristallographique. La transposition de la technique du recuit physique à la métaheuristique d'optimisation du recuit simulé est basée sur les analogies suivantes : La fonction objectif à optimiser est assimilée à l'énergie du matériau et la température est représentée par un paramètre de contrôle définissant le schéma de refroidissement. L'idée fondammentale de cette méthode est qu'elle permet des mouvements qui mènent vers de mauvaises solutions par rapport à la solution courante dans le but de s'echapper des minima locaux.

Algorithm 7 Pseudo code de l'Algorithme du Recuit Simulé

Initialiser la température initiale T
Générer une solution initiale s
while la condition d'arrêt n'est pas vérifiée **do**
    Générer un voisin $s^{'}$ de s
    Evaluer $s^{'}$
    if $f(s^{'}) \leq f(s)$ **then**
        Remplacer s par s^{*}
    else if $e^{-\Delta f/T} > rand(0,1)$ **then**
        Remplacer s par s^{*}
    end if
    Mettre à jour la température T
end while
Retourner s

La probabilité d'accepter ce genre de solutions décroit au cours du processus de recherche. Le pseudo code de cette méthode est présenté ci dessus. Cet algorithme commence par générer une solution initiale (aléatoirement ou en utilisant une heuristique) et par initialiser un paramètre qui représente la température T. Ensuite, à chaque itération une solution $s^{'} \in V$ (qui représente l'ensemble des voisins de s). L'acceptation de $s^{'}$ comme nouvelle solution dépend de $f(s)$, $f(s^{'})$ et T. La solution $s^{'}$ remplace donc la solution courante s

si $f(s^{'}) < f(s)$ ou dans le cas où $f(s^{'}) \geq f(s)$ avec une probabilité qui depend de T et $f(s^{'}) - f(s)$. La probabilité est généralement calculée selon la distribution de Boltzmann $\exp(-\frac{f(s^{'})-f(s)}{T})$.

2.4.4 La méthode GRASP

GRASP est un acronyme pour greedy randomized adaptive procedures (La procédure de recherche gloutonne aléatoire adaptative) est une métaheuristique introduite par Feo et Resende en 1989 [57,58]. Elle est parmis les métaheuristiques les plus efficaces pour résoudre des problèmes d'optimisation combinatoire. Son fonctionnement se base sur la répétition de deux phases : Une construction gloutonne suivit par une recherche locale. La phase de construction construit une solution réalisable et si elle n'est pas réalisable, une procédure de réparation doit être appliquée pour atteindre la réalisabilité. Si la réalisabilité n'est pas atteinte cette solution est negligée et une nouvelle solution est construite. Une fois une solution réalisable est construite, son voisinage sera exploré dans la phase de la recherche locale dans le but de l'améliorer et de trouver un minimum local. Après un nombre donné d'itérations, l'algorithme GRASP se termine et la meilleure solution trouvée est conservée.

Algorithm 8 **Pseudo code de l'Algorithme de GRASP**

```
Initialiser le nombre maximum d'itérations
while le nombre maximum d'itérations n'est pas atteint do
    Construire une solution réalisable s
    Appliquer recherche local sur s
    Affecter à s la meilleure solution trouvée s*
end while
Retourner s
```

La phase de construction se caractérise par trois caractéristiques principales :

1. **Une fonction gloutonne :** Elle est utilisée pour déterminer quel élément candidat doit être ajouté à la solution construite partiellement. Cette fonction mesure le bénéfice d'inclure un élément donné dans la solution.

2. **Sélection aléatoire :** Tous les éléments qui peuvent être sélectionnés pour être inclus dans la solution sont classés dans une liste selon leur valeurs de la fonction gloutonne. Ensuite, un élément est sélectionné parmi les meilleurs candidats de la liste. La liste des meilleurs candidats est appelée la liste RCL (restricted candidate list).

3. **Qualité adaptive :** La procédure est adaptative car le bénéfice (valeur de la fonction gloutonne) de chaque élément est mis à jour à chaque itération de la phase de construction afin d'intégrer les nouvelles informations qui sont recueillies après la sélection de l'élément précédent.

Les solutions générées par la phase de construction de GRASP ne sont pas forcément des minima locaux. Ainsi la deuxième phase d'amélioration est necéssaire en utilisant la recherche locale ou une autre technique pour améliorer la solution fournie par la phase de construction.

2.4.5 La recherche à voisinage variable

La recherche à voisinage variable, Variable Neighborhood Search (VNS) Mladenovic en 1995 [59], Mladenovic et Hansen en 1997 [8], applique une stratégie basée sur le changement dynamique des structures de voisinages.
L'étape d'initialisation commence par définir un ensemble de structures de voisinages. Souvent une séquence d'ensemble de voisinages avec une cardinalité croissante est définie. Ensuite, une solution initiale est générée, l'indice du voisinage est initialisé et l'algorithme itère jusqu'à ce qu'une condition d'arrêt soit atteinte.

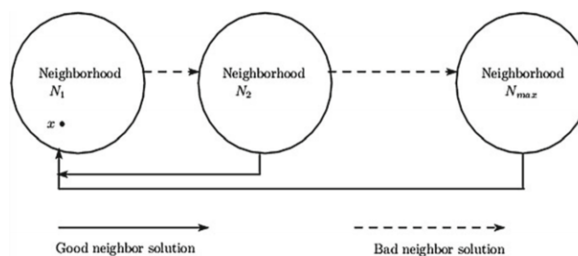


FIGURE 2.5 – Illustration du fonctionnement du VNS [11].

Algorithm 9 Pseudo code de l'algorithm de la recherche à voisinage variable [8]

Définir un ensemble de structures de voisinages V_k, $k = 1, \ldots, k_{max}$
Générer une solution initiale.
while Le nombre maximum d'itérations n'est pas atteint **do**
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    Choisir une solution $s^{'}$ du k^{me} voisinage de s
    $s^{''}$ est resultat d'une recherche locale en démarrant par $s^{'}$
    if $f(s^{''}) < f(s)$ **then**
      $s \leftarrow s^{''}$
      $k \leftarrow 1$
      Sinon $k \leftarrow k + 1$
    end if
  end while
end while

2.5 Les métaheuristiques à population de solutions

les métaheuristiques à base d'une population de solutions (les P-métaheuristiques). Elles manipulent non pas une seule solution, mais un ensemble de solutions à chaque itération. Ce sont des méthodes qui font évoluer simultanément un ensemble d'individus (solutions) dans l'espace de recherche d'une itération à une autre d'une manière itérative. L'intérêt est d'utiliser la population comme un facteur de diversité.

Plusieurs types de métaheuristiques à base de population existent dans la litérature selon la metaphore dont ils s'inspirent et qui a conduit à leur création, on peut citer les algorithmes évolutionnaires [60] inspirés de la génétique et de la théorie de l'évolution de C. Darwin et qui regroupe beaucoup d'algorithmes tels que l'algorithme génétique (GA) [65], l'algorithme à évolution différentielle (DE) [74], l'algorithme d'optimisation basé sur la biogéographie (BBO) [104], l'algorithme de la stratégie d'évolution (ES), les algorithmes d'intelligence en essaim tels que l'algorithme des colonies de fourmis (ACO) [52], les algorithmes à essaims de particules (PSO) [73], l'algorithme de la colonie d'abeilles (ABC) [77]. Aussi nous avons les algorithmes basés sur la physique tels que l'algorithme de recherche basé sur la galaxie (GbSA) [87], la recherche de système chargé (The charged system search) (CSS) [105] et l'algorithme de recherche gravitationnelle (Gravitational search algorithm) (GSA) [106]. La liste des métaheuristiques à population est très longue, dans ce qui suit nous donnons un bref aperçu sur quelques P-métaheuristiques.

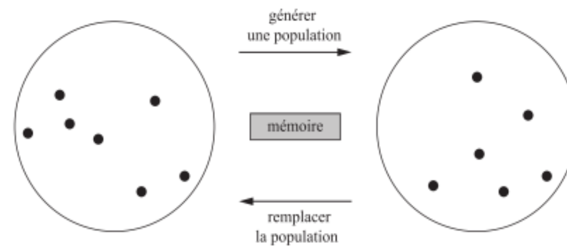


FIGURE 2.6 – Illustration du fonctionnement d'une métaheuristique à base de population (P-métaheuristique) [11].

2.5.1 L'algorithme génétique

Les algorithmes génétiques appatiennent aux algorithmes évolutionnaire parcequ'il s'inspire de la théorie d'évolution de Darwin. L'algorithme a été proposé par J.Holland 1975 [51,65], mais sa popularité revient à la parution du livre de D.E.Goldberg 1989 [107]. L'algorithme génére une population de solution à chaque itération en utilisant des individus (solutions) de la population de l'itération précedente et en appliquant sur ses derniers des opérateurs de sélection et de variation. Les opérateurs de sélection permettent à un individu d'être choisi pour la reproduction (croisement). Les opérateurs de variation peuvent être classés en deux catégories opérateurs de croisement et opérateurs de mutation. Les opérateurs de croisement s'appliquent sur chaque deux solutions (parents) séléctonnés de la population et produisent une ou plusieurs nouvelles solutions (enfants). Les opérateurs de mutation, apportent des modifications à une seule solution (individu) et génèrent une nouvelle solution (individu). Un algorithme génétique consiste donc à appliquer itérativement un ensemble d'opérations définies par l'algorithme10.

Algorithm 10 **Pseudo code de l'algorithm génétique**

```
Générer une population initiale aléatoire d'individus.
Evaluer les individus de la population initiale.
while le nombre maximum d'itérations n'est pas atteint do
    Sélectionner les meilleurs individus pour être utilisés.
    Générer de nouveaux individus en utilisant le croissement et la mutation.
    Evaluer les nouveaux individus.
    Remplacer les mauvais individus de la population par les meilleurs.
end while
```

2.5.2 L'algorithme à essaims de particules

L'optimisation par essaim de particules (OEP) (Particle Swarm Optimization (PSO), en anglais) est une métaheuristique d'optimisation qui dérive des algorithmes d'intelligence en essaim La classe de métaheuristiques inspirés du comportement collectif de certaines espèces, telles que les fourmis, les abeilles, les poissons, les oiseaux, etc. L'algorithme à essaims de particules a été introduit en 1995 par Kennedy et Eberhart [108]. Les individus de l'algorithme sont appelés particules et la population est appelée essaim. Dans cet algorithme, les solutions candidates (appelées particules) se déplacent dans l'espace de recherche en modifiant leur position et leur vitesse selon des règles simples, le mouvement de chaque particule est influencé par sa meilleure position connue et par la meilleure solution connue de l'essaim dans l'espace de recherche. Les positions des particules améliorées localement permettent à l'essaim de se déplacer globalement vers de bonnes solutions.
Chaque particule est donc caractérisée par un vecteur position et un vecteur de vitèsse qui dirige le mouvement de la particule dans l'éspace. Elle garde aussi en mémoire la méilleure position croisée et la meilleure position atteinte par l'essaim. Le vecteur vitesse de la particule prend en cosidération toute ses informations, qui sont mises à jour à chaque itération.

Algorithm 11 Pseudo code de l'algorithme à essaims particulaires

---

Initialiser l'essaim (positions et vitesses des particules).
Evaluer les particules.
while le nombre maximum d'itérations n'est pas atteint **do**
    for chaque particule de l'essaim **do**
        Mettre à jour la vitesse de la particule.
        Mettre à jour la position de la particule.
        Mettre à jour la meilleure position connue de la particule.
    end for
end while
Renvoyer la meilleure position connue de l'essaim.

2.6 L'hybridation des métaheuristiques

La majorité des métaheuristiques publiées dans la littérature sont d'une façon ou d'une autre hybrides. Le théorème du no free lunch (Wolpert et Macready en 1997) [109] indique qu'une métaheuristique ne peut prétendre être plus efficace qu'une autre sur tous les problèmes possibles. Néanmoins, en l'implémentant et en la paramétrant d'une certaine façon,

elle peut être plus adaptée à certaines classes de problèmes.
Le principe donc des métaheuristiques hybrides est de combiner des métaheuristiques avec d'autres techniques d'optimisation. Ces techniques peuvent être d'autres métaheuristiques, des heuristiques, des méthodes exactes, etc. dans le but de créer des algorithmes plus performants. Plusieurs manières d'hybrider les métaheuristiques peuvent être envisager selon la nature du problème. Le premier livre consacré entièrement au sujet d'hybridation des métaheuristiques a été publié par Blum et Roli en 2008 [110].

2.6.1 Classification hiérarchique des méthodes hybrides

La Classification hiérarchique des méthodes hybrides a été proposé par Jourdan et al. [111]. Elle les classe selon le niveau qui peut être soit bas (Low-Level) soit haut (HighLevel) et le mode de l'hybridation qui peut être soit le mode relais ou le mode co-évolutionnaire.
Dans le niveau bas, une métaheuristique remplace un opérateur d'une autre méthode qui l'englobe. Par contre, dans le niveau haut de l'hybridation, chaque métaheuristique garde sa structure entière et fonctionne de manière indépendante.
L'hybridation à bas niveau a pour but d'utiliser les propriétés de diversification de la métaheuristique de base et d'intensifier la recherche avec l'hybridation embarquée.
L'hybridation à haut niveau consiste à combiner une métaheuristique avec d'autres méthodes sans que leur fonctionnements interne ne soient en relation. Le but est d'exécuter une séquence de méthodes. On dit que l'hybridation est séqentielle si les méthodes hybridées s'éx-

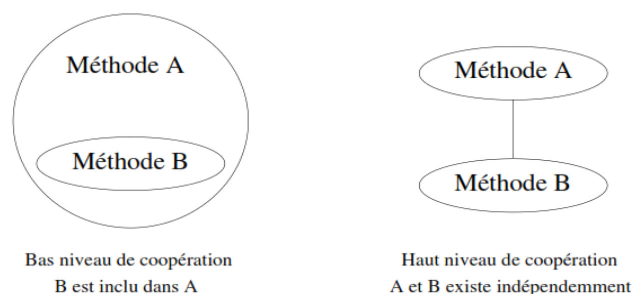


FIGURE 2.7 – Les différents niveaux d'hybridation

cecutent l'une après l'autre et le résultat de la méthode excécutée est communiqué (comme entrée) à la méthode qui s'éxcute juste après. Quand les différentes méthodes fonctionnent en parallèle pour explorer l'espace de recherche, on parle de mode coévolutionnaire. Ces classes sont comme suit :

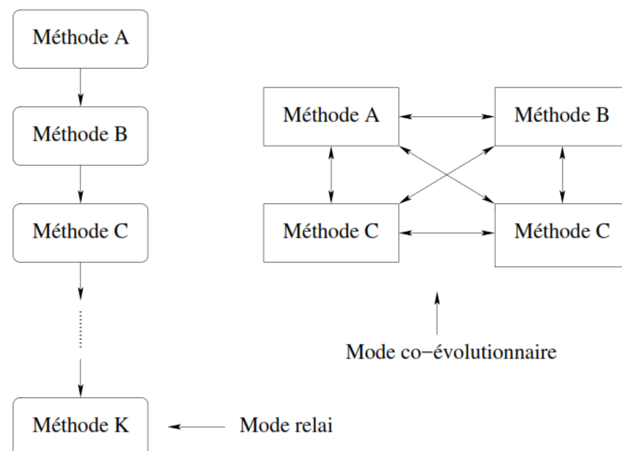


FIGURE 2.8 – Les différents modes d'hybridation

2.6.1.1 L'hybridation relais de bas niveau (BNR)

Low-level Relay Hybrid method (LRH), Dans cette classe on regroupe les méthodes hybrides qui sont formées par un Algorithm maitre et un algorithm esclave c'est à dire une des méthode sera la principale et l'autre sera incorporée dans cette dernière. Le fonctionnement donc la méthode principale dépend de la méthode incorporée qui doit lui communiquer son résultat pour pouvoir fonctionner. comme exemple on peut citer celui de Martin et Otto [112] qui ont inséré la méthode de descente dans un algorithme de recuit simulé.

2.6.1.2 L'hybridation co-évolutionnaire de bas niveau (BNC)

Low-level Teamwork Hybrid methode (LTH). Dans cette classe la méthode incorporée doit pouvoir être exécutée en parallèle avec la méthode globale. L'avantage de ce type d'hybridation est de compenser la puissance d'exploitation d'une recherche locale et celle d'exploration d'une recherche globale. Stûtzle et Hoos [113] incorporent une fonction de recherche locale dans un algorithme de colonie de fourmis pour résoudre le problème du voyageur de commerce et celui de partition de graphes. Cotta et al [114] ont incorporée une méthode de branch and bound dans un algorithme génétique, la méthode de branch and bound a joué le role d'un opérateur de croissement.

2.6.1.3 L'hybridation relais de haut niveau (HNR)

High-level Relay Hybrid (HRH). Souvent utilisée et consiste à hybrider des métaheuristiques qui fonctionnent de manière séquentielle c'est-à-dire la ou (les) solution(s) finale(s) de la première métaheuristique représente la ou (les) solution(s) initiale(s) de la métaheuristique

suivante. Dans cette procédure, toutes les méthodes gardent leur intégrité. Dans cette classe on retrouve plusieurs méthodes dans la littérature notamment celles qui hybrident une métaheuristique à population avec une métaheuristique à solution unique, ou même l'utilisation d'une heuristique pour la construction de solution de démarrage pour une métaheuristique à solution unique. Lin, Kao et al. [115] ont proposé une hybridation où la méthode du recuit simulé crée une population initiale pour un algorithme génétique.

2.6.1.4 L'hybridation co-évolutionnaire de haut niveau (HCH)

High level co-evolutionary Hybrid Method, cette classe contient des algorithme formés d'hybridation de méthodes qui ne sont pas incorporées les unes dans des autres, et qui fonctionnent en parallèles. V. Nwana et al. [116] ont proposé une méthode hybride formée d'un algorithme de recuit simulé et de la méthode de branch and bound où les deux méthodes hybridées fonctionnent en parallèle.

Dans ce chapitre quelques approches métaheuristiques ont été présentées, nous présentons dans le chapitre suivant une nouvelle approche métaheuristique que nous avons dévelopé.

Science is not a steady march from ignorance to knowledge. It's more like mountaineering expedition. On the way up an unscaled peak, climbers will gain some altitude on one route, then find it's a dead end. They'll spot a better one, backtrack a little and move on. The fact that they sometimes have to take a step backward for every two steps forward doesn't mean they are wasting their time. It means that inching up an uncharted mountain is tough work. When you step back, though, and take a look at the overall picture , a long view from the upper slopes of the mountain, it turns out in hindsight that the path was clear.

Michael D. Lemonick, Science Writer

CHAPITRE 3

DÉFINITION D'UNE NOUVELLE APPROCHE MÉTAHEURISTIQUE : LA MONARCHIE METAHEURISTIQUE

La métaheuristique proposée est la Monarchie Métaheuristique (MN) [13], c'est une métaheuristique inspirée d'une forme de gouvernement appelée monarchie. Une monarchie est une forme de gouvernement dans laquelle le pouvoir suprême est entre les mains d'un individu, qui est le chef de l'État, jusqu'à sa mort ou son abdication. Le chef d'une monarchie s'appelle un monarque ou un roi. C'était une forme de gouvernement commune dans le monde entier à l'époque antique et médiévale. De nos jours, plus de quarante nations souveraines dans le monde ont des monarques agissant en tant que chefs d'États. Les monarchies peuvent être classées selon le mode de sélection du monarque, par exemple une monarchie élective est une monarchie gouvernée par un monarque élu, par contre pour la monarchie héréditaire, le trône est transmis comme un héritage familial, mais les monarchies électives peuvent se transformer en des monarchies héréditaires au fil du temps et celles qui sont héréditaires peuvent parfois avoir des aspects électifs.

3.1 Définitions nécessaires pour comprendre la méthode proposée

3.1.1 Monarchie héréditaire

La monarchie héréditaire est le style de monarchie le plus répandu ; c'est la forme utilisée par presque toutes les monarchies existantes dans le monde. Sous une monarchie héréditaire, tous les monarques sont issus de la même famille et le trône est transmis d'un membre

vers un autre. Le système héréditaire présente les avantages de stabilité, de continuité et de prévisibilité.

3.1.2 Dynastie

Une dynastie (du grec dunasteia, lui-même dérivé de dunastes, qui signifie, dirigeant ou officier) est une famille ou une lignée de dirigeants, une succession de souverains d'un pays appartenant à une même famille ou qui retracent leur descendance à un ancêtre commun. Les personnes en ligne pour devenir monarques sont appelées dynastes.

3.1.3 L'ordre de succession

Un ordre de succession ou la ligne de succession est la séquence des personnes éligibles pour obtenir le trône. Dans une monarchie héréditaire, la position de monarque est héritée selon un ordre de succession établi, ce qui permet de remplacer immédiatement le monarque après une vacance imprévue. Tous les monarques possibles sont légalement reconnus comme nés ou descendants de la dynastie.
Différents systèmes de succession ont été utilisés pour sélectionner les monarques suivants, tels que la primogéniture, la succession adelphique et la selection semi elective. Pour la primogéniture, l'aîné des enfants du monarque est le premier à devenir monarque. Dans certaines monarchies, la succession au trône utilise un autre type de succéssion qui est la succession adelphique qui donne d'abord le trône au prochain frère aîné du monarque. La méthode de sélection semi-élective qui donne du poids au mérite pour choisir le prochain monarque. Une monarchie autoproclamée est établie lorsqu'une personne revendique la monarchie sans aucun lien historique avec une dynastie antérieure.

3.2 Description de la monarchie metaheuristique

La monarchie métaheuristique inspirée du système de gouvernement monarchique commence d'abord par trouver deux solutions au problème c'est deux solutions peuvent être générées aléatoirement ou en utilisant des heuristiques, l'une de ces deux solutions représente le Roi, et la seconde aidera à former la première liste dynastie contenant les eventuels successeurs au trône. Pour cette solution Roi (king), nous allons créer deux listes : la liste des enfants L_1 et la liste des proches (frères, cousins,...) L_2. La liste de la dynastie sera alors D, l'union de L_1 et L_2.
Le prochain Roi sera choisi en utilisant l'un des trois modes de succession retenus parmis

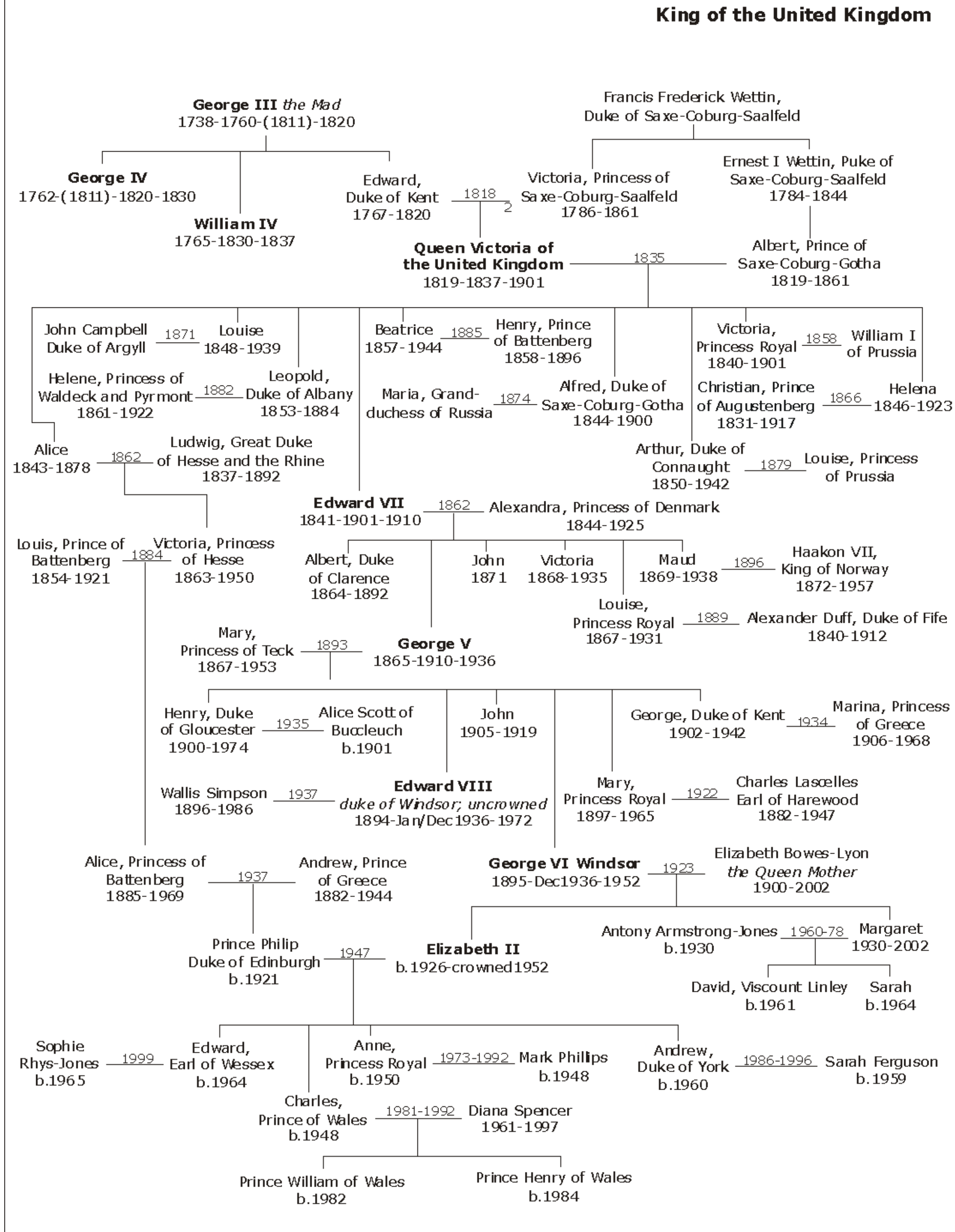


FIGURE 3.1 – Exemple de dynastie de la monarchie de la grande bretagne.

l'ensemble des modes de succession existants qui sont : primogéniture, la succession adelphique et la méthode de sélection semi-élective. Nous avons ajouté une quatrième méthode de sélection nommée l'intrus qui représente la situation de la monarchie autoproclamée, ça signifie que le prochain roi n'est pas choisi de la liste de la dynastie. L'algorithme continue d'une itération à une autre, chaque itération commence par une nouvelle solution qui représente le Roi à laquelle nous associons une liste de dynastie, cette liste est mise à jour à chaque itération. L'algorithme se termine lorsqu'il atteint un nombre d'itérations prédéfini.

3.2.1 Trouver le premièr Roi

La première solution Roi peut être générée de manière aléatoire ou créée en utilisant une heuristique appropriées au problème traité.

3.2.2 Creation de la liste L_1

Chaque solution de la liste L_1 est obtenue en appliquant un croisement entre la solution Roi et une autre solution (trouvé par une heuristique constructive, ou généré aléatoirement). Il existe de nombreux types de méthodes de croisement, mais il est essentiel que nous utilisions et appliquions une méthode de croisement produisant des résultats valides (valables) pour notre problème.
Une fois les deux solutions parents nécessaires à chaque debut d'itération sont formées qui sont La solution Roi et l'autre solution qui aide à créer les éléments de la liste L_1, de nouvelles solutions candidates seront créées en appliquant une recombinaison (crossover). Le croisement combine des sous-ensembles de solutions parents en échangeant certaines de leurs composantes. Ci dessous nous présentons quelques types de croissement qui existent dans la litérature. La nature du croisement à utiliser dépend du problème traité et de la manière dont les solutions parents sont représentées.

3.2.2.1 L'opérateur de croisement OX (Order Crossover (OX)) [2]

L'opérateur de croisement OX choisit deux points aléatoires de croisement. L'enfant hérite les éléments situés entre les deux points de croisement, tirés du premier parent. Ces éléments occupent les mêmes positions, et donc apparaissent dans le même ordre que dans ce dernier. Les éléments restants sont hérités du deuxième parent dans l'ordre dans lequel ils apparaissent dans ce parent, en commençant par la première position suivant le deuxième point de croisement et en omettant tous les éléments déjà présents dans l'enfant.

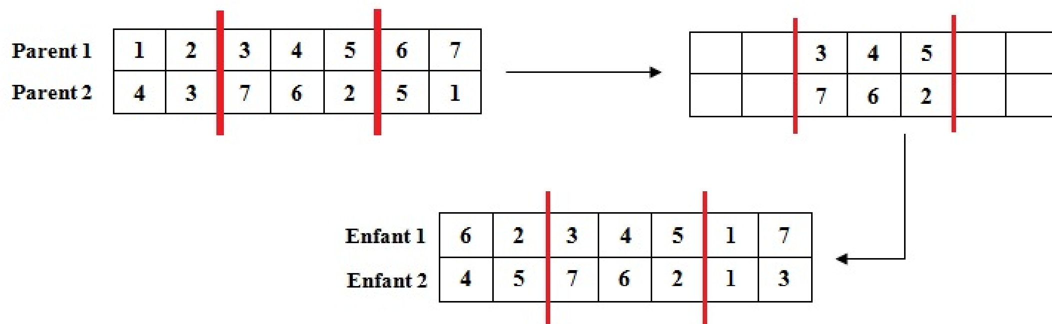


FIGURE 3.2 – Illustration du Order Crossover (OX).

Dans l'exemple illustré par la Figure 3.2, l'enfant hérite du premier parent les éléments (3, 4 et 5) qui sont situés entre les deux points de croisement. Ensuite, aprés le deuxième point de croisement nous le complétons avec les éléments qui manquent $(1,2,6,7)$ mais selon leur ordre d'apparition dans le deuxième parent, à partir de la position de (1) qui est le premier élément après le deuxième point de croisement dans le deuxième parent. Donc l'ordre sera $(1,7,6,2)$.

3.2.2.2 L'opérateur de croisement à un point (Single Point Crossover) [2, 3]

L'algorithme génétique traditionnel utilise un croisement en un seul point, où une position sur chaque parent est séléctionnée et les éléments après le site (point) de croisement sur chaque parent sont échangés entre les deux parents. Le point de croisement est choisi de manière aléatoire.

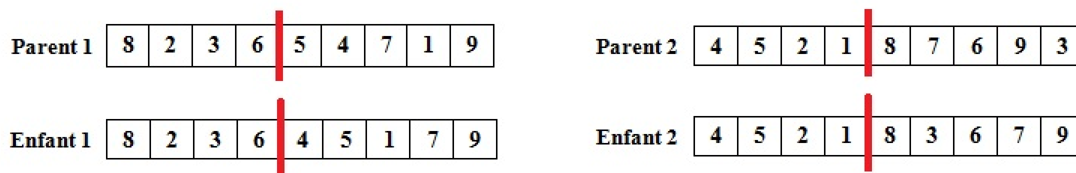


FIGURE 3.3 – Illustration du croisement en un point.

Pour construire le premier enfant, la première partie (8236) du premier parent est d'abord recopiée, puis ensuite les éléments (54719) de la deuxième partie de ce parent sont réordonnés dans l'ordre d'apparition qu'ils ont dans le deuxième parent. Le second enfant est généré de manière similaire, en inversant les rôles des deux parents.

3.2.2.3 L'opérateur de croisement à deux points (Two Point Crossover) [2, 3]

Dans le croisement à deux points, deux points de croisement sont choisis et les elements entre ces deux points sont échangés entre les deux parents.

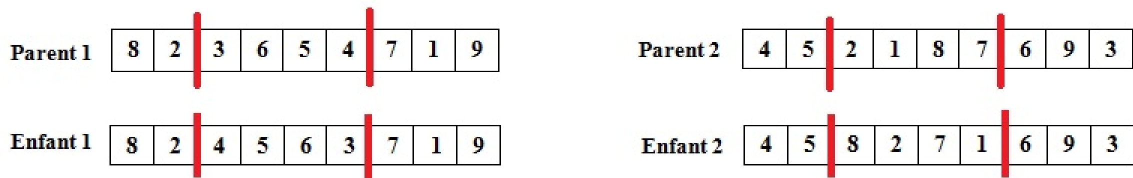


FIGURE 3.4 – Illustration du croisement en deux points.

Dans le processus qui construit le premier enfant sur la Figure3.4, les éléments $(8, 2)$ et $(7, 1, 9)$ des parties impaires du premier parent sont héritées de ce parent au premier enfant, et les éléments des parties paires $(3, 6, 5\ et\ 4)$ sont réordonnés dans le même ordre que dans l'autre parent. Pour le second enfant, les rôles des parents sont inversés.

3.2.2.4 L'opérateur de croisement uniforme (Uniform Crossover) (UX) [2]

Le croisement uniforme de permutations tend à faire hériter un enfant d'une combinaison des ordres existants dans deux parents. Le croisement se déroule en trois étapes :

- un masque binaire est engendré aléatoirement.
- deux parents sont appariés. Les (0) du masque binaire définissent les positions préservées dans la séquence du parent1, et les (1) du masque binaire définissent les positions préservées dans la séquence du parent2.
- pour obtenir l'enfant1, les éléments non préservés du parent1 sont permutés de façon à respecter l'ordre qu'ils ont dans le parent2.

Un exemple d'application de ce type de croisement est présenté dans la Figure 3.5.

3.2.3 Creation de la liste L_2

La liste L_2 est formée par des solutions voisines à la solution Roi. Plusieurs types de voisinages peuvent être utilisés, le choix des voisinages utilisés est relatif au problème traité. Ci dessous nous présentons quelques types de voisinages qui exitent dans la litérature. Rappelons d'abord la notion de voisinage :

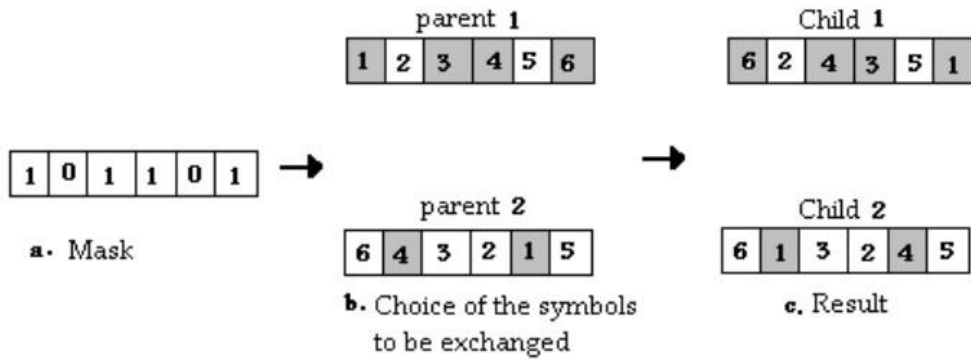


FIGURE 3.5 – Illustration du croisement Uniforme (UX).

On définit le voisinage d'une solution à l'aide d'une transformation élémentaire ou locale. On appelle transformation élémentaire toute opération permettant de changer faiblement la structure de la solution à laquel on l'applique. Une fois la transformation élémentaire choisie, on peut définir le voisinage $V(x)$ d'une solution x, par l'ensemble des solutions que l'on peut obtenir en appliquant à x cette transformation locale.

Dans ce qui suit nous présentons quelques voisinages qui peuvent être appliqués sur les problèmes d'optimisation combinatoire à permutation. Des voisinages simples peuvent être définis pour des problèmes où une solution est **une permutation**.
De nombreux problèmes d'optimisation combinatoire peuvent s'exprimer sous la forme d'une recherche de permutations : celui du voyageur de commerce (ordre dans lequel on parcourt les villes) celui de l'affectation quadratique (site de chaque unité à placer) ou encore celui de la chaîne de montage (ordre dans lequel on fait passer les travaux sur la chaîne).
On note par Ω l'ensemble de toutes les permutations possibles de taille n, une permutation $\pi \in \Omega$ est une bijection de l'ensemble des entiers $\{1, 2, \ldots, n\}$ sur lui même. Une permutation est un ordre d'éléments $\{1, 2, \ldots, n\}$ qui est $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ où $\pi(i) \in \{1, 2, \ldots, n\}$ est l'élément de la i^{me} position et $\pi(i) \neq \pi(j), \forall i \neq j$.

3.2.4 L'inversion adjacente (Adjacent swap) [4]

Soit une solution $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, ces voisins suivant ce type de voisinage résultent de l'inversion de deux éléments adjacents de cette solution.

$$\begin{aligned} V_{IA}(\pi(1), \pi(2), \dots, \pi(n))) = \ & \{(\pi^{'}(1), \pi^{'}(2), \dots, \pi^{'}(n)) \mid \pi^{'}(k) = \pi(k), \quad \forall k \neq i, (i+1) \\ & \pi^{'}(i) = \pi(i+1), \pi^{'}(i+1) = \pi(i)\}, \end{aligned}$$

Le nombre de voisins d'une permutation de taille n suivant ce voisinage est $(n-1)$.
Par exemple, dans l'espace des permutations de taille $n = 5$, l'ensemble des voisins de la solution $\pi = (1, 2, 3, 4, 5)$ est

$$V_{IA}(1, 2, 3, 4, 5) = \{(2, 1, 3, 4, 5), (1, 3, 2, 4, 5), (1, 2, 4, 3, 5), (1, 2, 3, 5, 4)\}.$$

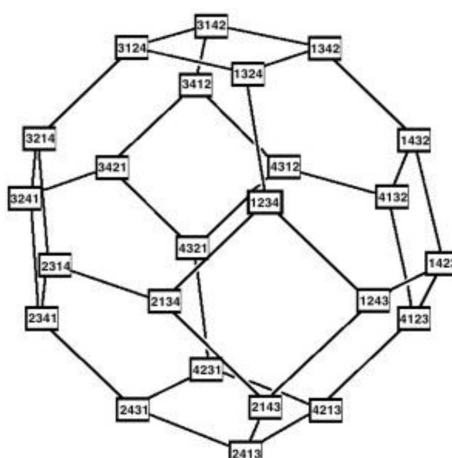


FIGURE 3.6 – L'espace des permutations de taille 4 connectées selon le voisinage adjacent swap [4]

3.2.5 L'inversion (Swap) [4]

L'inversion ou 2-point voisinage, ou 2- echange voisinage (2-exchange neighborhood) considère que deux solutions sont voisines si l'une est générée par l'inversion de deux positions de l'autre, les positions inversés ne sont pas nécéssairement adjacentes.

$$\begin{aligned} V_{I}(\pi(1), \pi(2), \dots, \pi(n))) = \ & \{(\pi^{'}(1), \pi^{'}(2), \dots, \pi^{'}(n)) \mid \pi^{'}(k) = \pi(k), \quad \forall k \neq i, j, \\ & \pi^{'}(i) = \pi(j), \pi^{'}(j) = \pi(i), i \neq j\}, \end{aligned}$$

Sous ce voisinage, une solution a $\frac{n(n-1)}{2}$ voisins.
Prenant la même permutation $\pi = (1,2,3,4,5)$ comme dans le cas précédent, l'ensemble des voisins selon ce type de voisinage est

$$\begin{aligned} V_I(1,2,3,4,5) = \; & \{(2,1,3,4,5),(3,2,1,4,5),(4,2,3,1,5),(5,2,3,4,1),(1,3,2,4,5),\\ & (1,4,3,2,5),(1,5,3,4,2),(1,2,4,3,5),(1,2,5,4,3),(1,2,3,5,4)\}. \end{aligned}$$

3.2.6 Voisinage 2-opt

Le but du voisinage connu sous le nom de 2-opt [117] consiste précisément à supprimer les croisements des arêtes dans une tournée du problème de voyageur de commerce. Si l'on nomme $\pi(i)$ la i-ème ville que l'on visite dans la solution courante ($i = 0,1,\ldots,n-1$), le voisinage 2-opt commence par choisir deux villes u et v ; ($v \neq u, u+1$). La solution voisine θ est obtenue à partir de $\pi(i)$ en supprimant les deux arcs $(u, u+1)$ et $(v, v+1)$, en ajoutant les arcs (u, v) et $(u+1, v+1)$ et en inversant le sens de parcours du chemin entre $u+1$ et v.

3.2.7 Sélection du prochain Roi

Quatre modes de sélection ont été utilisés qui sont comme suit :

3.2.7.1 Primogéniture

La primogéniture vient du latin "primo", premier, et "genitus", engendré. Dans le cas de la primogéniture, c'est l'aîné, le premier enfant du monarque qui monte sur le trône lors de la mort ou de l'abdication de ce dernier.
En l'absence d'enfant, les frères succédent au trône. Dans la Monarchie métaheuristique, si le mode de sélection du prochain Roi est primogéniture, le choix du prochain Roi se fera à partir de la liste L_1.

3.2.7.2 La succession adelphique

Au mode adelphique, dans ce mode horizontal de transmission du pouvoir, c'est l'accession au pouvoir du membre le plus puissant du lignage généralement un frère, un neveu ou un cousin plus ou moins proche du souverain décédé ou déposé qui est favorisé.
Dans ce type de de succession, l'ordre de succession au trône préfère le frère cadet du Roi aux fils du Roi. Les enfants d'un Roi (la génération suivante) ne succède au trône que lorsque la liste des hommes de la génération la plus âgée est complétement épuisée. Dans la monarchie

métaheuristique, si le mode de sélection du prochain roi est adelphique, le choix du prochain roi se fera à partir de la liste L_2.

3.2.7.3 Sélection semi-élective

La sélection semi-élective est une méthode de succession rare où le roi est choisi parmi la liste de la dynastie par élection. Dans la monarchie métaheuristique, si le mode de sélection du prochain roi est selon le mode semi-elective, le prochain roi sera la meilleure solution dans la liste D, qui est la liste de la dynastie contenant tous les membres de L_1 et L_2 listes.

3.2.7.4 L'intrus

Une monarchie autoproclamée est établie lorsqu'une personne revendique le trône sans aucun lien historique avec une dynastie antérieure. Dans la monarchie métaheuristique, si le mode de sélection est L'intrus, la solution du prochain roi ne sera pas choisie de la liste de la dynastie. Il sera crée en utilisant n'importe quel autre métaheuristique.

3.2.8 Le framework de la Monarchie Métaheuristique

Comme tout autre métaheuristique, la monarchie métaheuristique peut être définie par un framework générique. L'un des principaux avantages de cette métaheuristique est qu'elle peut être considérée comme une métaheuristique à solution unique si nous décidons de ne conserver que la solution Roi à chaque itération. Elle peut également être utilisée comme métaheuristique à population si toutes les solutions de la liste dynastie associée à la solution Roi est gardée. L'algorithme12 présente le framework de la Monarchy metaheuristic.

3.2.9 le paramètre de la monarchie métaheuristique

La monarchie métaheuristique (MN) nécessite un seul paramètre qui est la taille des deux listes L_1 et L_2. Plusieurs manières sont possibles pour définir ce paramètre.
L'utilisateur peut choisir de définir la taille de ces deux listes au début de l'algorithme, puis à chaque itération et pour chaque solution Roi, la taille des deux listes L_1 et L_2 sera la même. Une autre manière qui donne plus d'importance à la qualité de la solution choisie comme solution Roi à chaque itération consiste à choisir la taille des deux listes en fonction du pourcentage d'amélioration de la solution. La taille des listes peut être changée d'une itération à l'autre, et la taille de L_1 peut également être différente de celle de L_2.

Algorithm 12 Pseudocode de la Monarchie Métaheuristique

Initialisation

- Construire la première solution Roi K_1 aléatoirement ou en utilisant une heuris tique connue.
- Initialiser LK avec K_1. LK est la liste qui contient toutes les solutions Roi trouvées à toutes les itérations.
- Fixer les tailles initiales des deux listes L_1 et L_2.
- Générer L_1, L_2 associées à K_1.
- Initialiser $delta \leftarrow 0$ ($delta$ est le pourcentage d'amélioration de la solution).
- Fixer le nombre maximum d'iterations ($MaxIterations$).
- Initialiser le compteur d'itérations $N \leftarrow 1$.

while ($N < MaxIterations$) **do**

a) Choisir un mode de sélection de la prochaine solution Roi.

if (le mode de sélection est par Primogéniture) **then**

- Evaluer chaque solution existante dans L_1.
- Sélectionner la prochaine solution Roi à partir de la liste L_1.

else if (le mode de sélection est adelphique) **then**

- Evaluer chaque solution existante dans L_2.
- Sélectionner la prochaine solution Roi à partir de la liste L_2.

else if (le mode de sélection est semi-elective) **then**

- Construire la liste dynastie D telle que $D = L_1 \cup L_2$.
- Evaluer chaque solution existante dans D.
- Sélectionner la prochaine solution Roi à partir de la liste D.

else if (le mode de sélection est l'intrus) **then**

- Générer la prochaine solution Roi en utilisant n'importe quelle métaheuristique existante déjà dans la littérature.

end if

b) Calculer $delta$ le pourcentage d'amélioration de la solution.

if (Il n'y a pas eu une amélioration) **then**

1. Utiliser une des alternatives suivantes :

Alternative 1 :

- Mettre la solution Roi courante dans la liste LK et génerer L_1, L_2.
- La taille de L_1, L_2 diminuera en fonction de la valeur de $delta$.

Alternative 2 :

- Prendre la meilleure solution Roi trouvée au cours du processus de recherche et utiliser ses listes L_1, L_2.

2. $N \leftarrow N + 1$.

else if (Il y a eu une amélioration) **then**

1. Mettre la solution Roi dans la liste LK.

2. La taille de L_1, L_2 augmentera en fonction de la valeur de $delta$.

3. Générer L_1 et L_2.

4. $N \leftarrow N + 1$.

end if

end while

Critères d'arrêt : L'algorithme s'arrête lorsqu'il atteint MaxIterations.

Finalisation : Retourner la meilleure solution Roi obtenue.

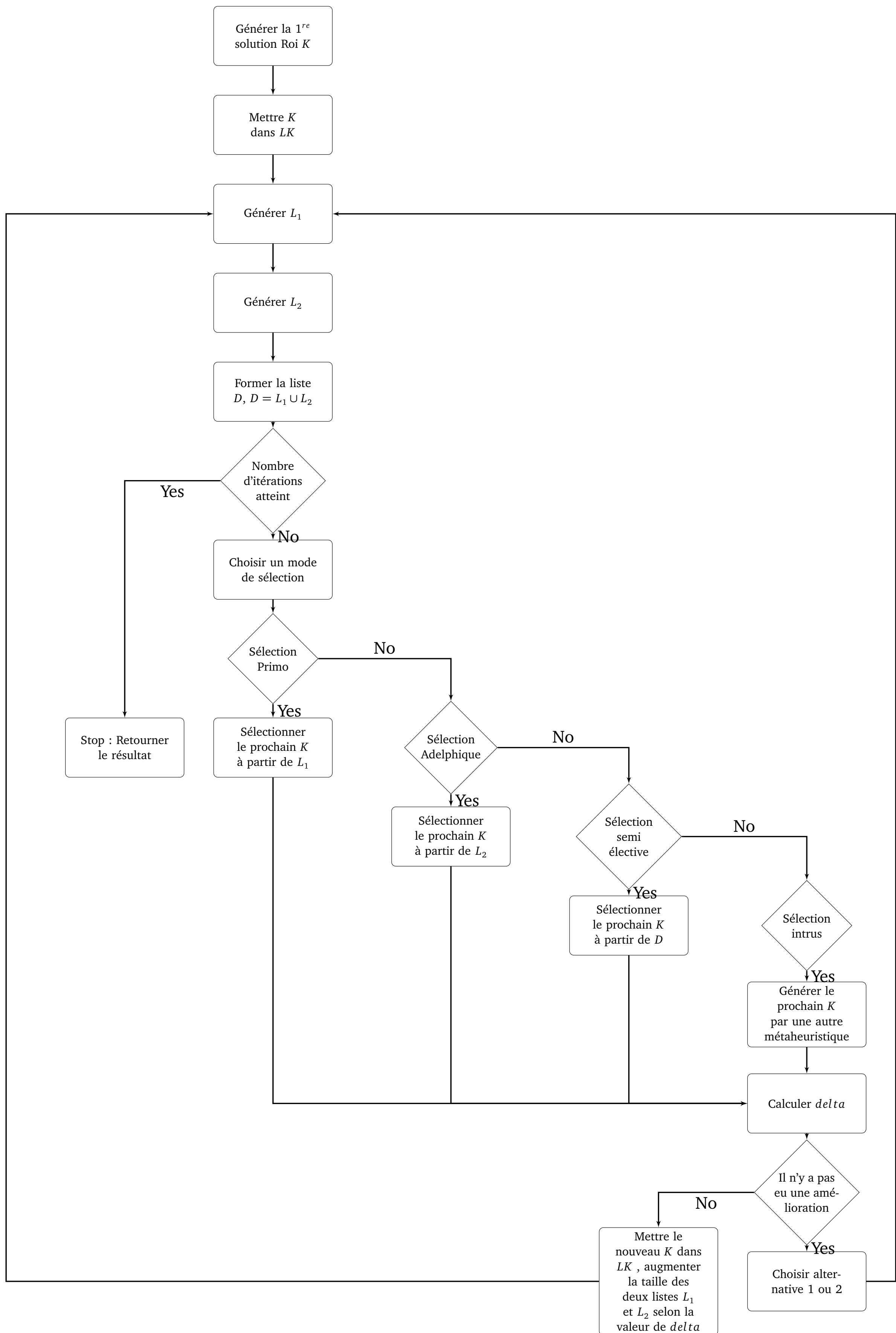


FIGURE 3.7 – Organigramme de la Monarchie Métaheuristique

3.2.10 Et si la prochaine solution Roi n'apporte pas d'amélioration ?

Nous avons testé deux alternatives au cas où la solution sélectionnée pour être la prochaine solution Roi n'apporterait aucune amélioration : La première consiste à conserver cette solution et la taille de L_1, L_2 construites pour cette solution diminuera en fonction de la valeur du pourcentage d'amélioration, cette alternative a été notée MN $_1$. La seconde consiste à négliger cette solution, revenir à la meilleure solution Roi trouvée au cours du processus de recherche et utiliser la liste dynastie de cette meilleure solution pour choisir la prochaine solution Roi. Cette alternative a été notée MN $_2$.

Dans ce chapitre une nouvelle approche métaheuristique intitulée "la Monarchie Métaheuristique" a été présentée, dans le chapitre suivant une première variante de cette métaheuristique sera présentée.

Nos plus grandes difficultés sont nos plus grandes occasions.

Sri Aurobindo

CHAPITRE 4 UNE PREMIÈRE VARIANTE DE LA MONARCHIE METAHEURISTIQUE APPLIQUÉE POUR LA RESOLUTION DU PROBLÈME DE VOYAGEUR DE COMMERCE

Une première variante de la Monarchie Metaheuristique [13] a été réalisée pour la résolution du problème de voyageur de commerce, les résultats de cette méthode comparés avec d'autre méthodes heuristiques qui existent dans la litérature prouvent que la méthode est compétitive avec les méthodes qui existent.

4.1 Le problème traité : Le problème du voyageur de commerce (TSP)

Nous avons utilisé le problème de voyageur de commerce pour analyser et tester notre méthode. Il est souvent utilisé pour tester les performances et le comportement des méta-heuristiques. C'est l'un des problèmes d'optimisation combinatoire les plus étudiés. Le TSP est très facile à décrire, mais difficile à résoudre. C'est un problème NP-difficile ; Cela signifie qu'une méthode garantissant une solution optimale pour toutes les instances de ce problème dans un délai d'exécution raisonnable n'existe pas encore.

Lorsqu'il pose le problème pour la première fois en 1857, William Rowan Hamilton énonce : " Un voyageur de commerce doit visiter une et une seule fois un nombre fini de villes et revenir à son point d'origine. Trouvez l'ordre de visite des villes qui minimise la distance totale parcourue par le voyageur ". Le problème du voyageur de commerce est connu

également dans sa version anglo-saxonne par Traveling Salesman Problem (TSP).

Les données d'un problème TSP sont donc un ensemble de N villes et les distances qui les séparent, le voyageur a pour tâche de visiter toutes les villes et doit passer par chaque ville une et une seule fois, de sorte que la durée totale de la tournée soit minimale.

Les solutions sont codées en utilisant la représentation des chemins (path representation) qui est la représentation la plus naturelle qui représente une tournée sous forme d'une liste de N villes. Si la ville j est le k^{me} élément de la liste cela veut dire que la ville j sera la k^{me} ville à visiter.

4.2 Application de la Monarchie Metaheuristique au problème TSP

La Monarchy Metaheuristic commence par choisir une stratégie de création de la première solution Roi, ensuite les techniques utilisées pour la création des listes L_1, L_2 et enfin choisir une métaheuristique pour la procédure Intru de sélection.

4.2.1 Création de la première solution Roi

La première solution Roi pour le TSP a été crée à l'aide d'une heuristique constructive basée sur l'algorithme du plus proche voisin, il s'agit de l'un des premiers algorithmes utilisés pour déterminer une solution au TSP. Un algorithme glouton est basé sur le paradigme : à chaque étape, il fait le choix le plus "intéressant" à cet instant. Dans le cas du TSP l'algorithme commence par une ville choisie au hasard, tant qu'il existe des villes qui n'ont pas encore été visitées, il visite la ville la plus proche. Une fois que l'algorithme aura terminé de visiter toutes les villes, il s'arrêtera. Cet algorithme trouve rapidement une tournée, mais cette dernière n'est souvent pas la meilleure. le pseudocode de cet algorithme est présenté par l'algorithme 13.

4.2.2 Creation de la liste L_1

Comme c'est déjà expliqué dans le chapitre précédent, pour former la liste L_1, on doit avoir deux solutions de démarrage, la première solution Roi est déjà trouvée à partir de la première étape par l'heuristique gloutonne, la deuxième solution peut être générée aléatoire-

Algorithm 13 Algorithme Glouton

Input C : ensemble de villes à visiter, $startpos$: ville.
Output V : une tournée.
Begin
$currentpos \leftarrow startpos$
$V \leftarrow \{\}$
while $|V| < |C|$ **do**
    $nextpos \leftarrow min_c dist(currentpos, c), \forall c \in C \setminus V$
    $V \leftarrow V \cup nextpos$
    $currentpos \leftarrow nextpos$
end while
End

ment ou en utilisant une heristique de construction de solution aussi. Une fois on a les deux solutions, on effectue le croisement. Puisque le problème traité est le TSP, il est essentiel de choisir et d'appliquer un type de croisement qui produit des solutions valides pour ce problème. Le type de croisement utilisé dans notre cas est le (OX) Order crossover, il a été proposé par Davis en 1985 [118], déjà expliqué dans le chapitre précédent par example, on consider les deux tournée parents suivantes :
$(2, 0, 1, 3, 4)$, $(0, 1, 3, 4, 2)$, deux points de croisement sont choisis (identiques pour les deux parents). Supposant que ces deux points se situent entre les positions deux et trois et entre trois et quatre : Les visites pour enfants qui en résulteront auront la même visite entre les points de coupure que les visites pour les parents, comme suit : $(2, 0|1|3, 4)$, $(0, 1|3|4, 2)$. Les tournées enfants qui resultent auront une sous-tournée identiques à leurs parents entre les deux points de croisement comme suit : $(**|1|**)$, $(**|3|**)$.
Ensuite, en utilisant l'autre tournée parent, en commençant par le deuxième point de croisement et en omettant les villes déjà placée dans la tournée enfant, les elements restants sont insérés dans le même ordre dans lequel ils apparaissaient dans cette tournée parent. Lorsque la fin de la tournée parent est atteinte, il continue depuis sa première position. Les tournées enfants qui résultent de cet exemple seront : $(0, 3|1|4, 2)$ and $(0, 1|3|4, 2)$.

4.2.3 Creation de la liste L_2

La liste L_2 est formée par les solutions voisines de la solution Roi, dans notre cas trois types de voisinages ont été utilisés : Le voisinage à un point, le voisinage à deux points, le voisinage 2-opt.

4.2.3.1 Le voisinage à un point

Etant donnée une tournée, nous sélectionnons une ville au hasard, puis nous la déplaçons à toutes les autres positions de la tournée. Par exemple, pour la tournée $(0,1,2,3,4)$ et la ville séléctionnée 2, les tournées qui résultent sont : $(2,0,1,3,4)$, $(0,2,1,3,4)$, $(0,1,3,2,4)$, $(0,1,3,4,2)$. le pseudocode de l'algorithme du voisinage à un point est présenté par l'algorithme 14.

Algorithm 14 Le voisinage à un point (↓ Tournée : une tournée, K : entier positif, pos : entier positif, ↑ Voisins : une liste de tournées)

Entrées↓ Tournée : une tournée, K, pos : entiers positifs.
Sortie↑ Voisins : une liste de K voisins.
Début
$j := 0$; Voisins:=Null ; ville:=Tournée.get(pos) ; //la fonction get(pos) donne accès à la ville ayant la position pos dans la tournée.
while $(j \leq K)$ **do**
  voisin:=Null ;
  if $(j \neq pos)$ **then**
    $i := 0$;
    while $i \leq (Tourne.size() - 1)$ **do**
      if $(i \neq pos)$ **then** voisin.set(i)=Tournée.get(i) ;
      end if
      $i := i + 1$;
    end while
    voisin.add(pos,city) ;
    // La fonction add(pos, city) insérera ville à la position pos de la tournée voisin.
  end if
  Voisins:=Voisins.add(voisin) ;
  $j := j + 1$;
end while
Fin

4.2.3.2 Le voisinage à deux points

Etant donnée une tournée, nous sélectionnons une ville au hasard, puis nous la permutons avec une autre ville de la tournée. Par exemple pour la tournée $(0,1,2,3,4)$ et la ville séléctionnée 2, les tournées qui résultent sont :
$(2,1,0,3,4)$, $(0,2,1,3,4)$, $(0,1,3,2,4)$,$(0,1,4,3,2)$. le pseudocode de l'algorithme du voisinage à deux points est présenté par l'algorithme 15.

Algorithm 15 Le voisinage à deux points (↓ Tournée : une tournée, K : entier positif, pos : entier positif, ↑ Voisins : une liste de tournées)

Entrées↓ Tournée : une tournée, K,pos : entiers positifs.
Sortie↑ Voisins : une liste de K voisins.
Début
$j := 0$; Voisins:=Null ;
while $(j \leq K)$ **do**
  voisin:=Null ;
  voisin:=Tournée.copy ;
  // La fonction copy copiera la liste des villes de la liste Tournée à la liste voisin.
  voisin.switch(pos,j) ;
  // La fonction switch(pos,j) permute entre les villes de positions pos et j.
  Voisins:=Voisins.add(voisin)
  $j := j + 1$
end while
Fin

4.2.3.3 Le voisinage 2-opt

Chaque voisin 2-opt est construit par l'algorithme bien connu 2-opt proposé par Croes en 1958 [117]. La transformation elementaire 2-opt (move) supprime deux arêtes, qui coupe la tournée en deux chaînes, et ensuite elle les joint avec l'autre manière possible. le pseudocode de l'algorithme du voisinage 2-opt est présenté par l'algorithme 16.

Algorithm 16 Le voisinage 2-Opt (↓ Tour : une tournée, pos1 : entier positif, ↑ Voisins : une liste de tournées

Entrées ↓ Tour : une tournée, pos1 : entiers positifs.
Sortie ↑ Voisins : une liste de voisins.
Début
$pos2 := pos1 + 1$; $i := 0$; Voisins:=Null ;
while $(i \leq Tour.size() - 3)$ **do**
  voisin:=Null ; voisin:=Tour.copy ;
  $pos3 := pos2 + 1 + i$;
  $pos4 := pos3 + 1$;
  //La fonction ReversePath(pos2, pos3) inverse le chemin entre pos2,pos3.
  voisin:=voisin.ReversePath(pos2, pos3) ;
  Voisins:=Voisins.add(voisin) ;
end while
Fin

4.2.4 Selection du prochain Roi

Quatre modes de sélection ont été utilisés qui sont :

4.2.4.1 Primogéniture

Le pseudocode du mode de selection Primogéniture est présenté par l'algorithm 17, si le mode de sélection du prochain Roi est primogéniture, le choix du prochain Roi se fera à partir de la liste L_1.

Algorithm 17 Procedure Primogéniture (↓ L_1 : une liste de tournées, ↑ T : une tournée)

Entrée ↓ L_1 : une liste de tournées.
Sortie ↑ T : la meilleure tournée de L_1.
Début
// BestFitness (↓ L_1 : une liste de tournées) est une fonction qui calcule la valeur de la fonction objectif de chaque tournée de L_1 et renvoie la tournée qui a la meilleure valeur.
T= BestFitness (L_1) ;
Fin

4.2.4.2 Sélection Adelphique

Le pseudocode du mode de selection Adelphique est présenté par l'algorithm 18, si le mode de sélection du prochain Roi est adelphique, le choix du prochain Roi se fera à partir de la liste L_2.

Algorithm 18 Procedure Adelphique (↓ L_2 : une liste de tournées, ↑ T : une tournée)

Entrée ↓ L_2 : une liste de tournées.
Sortie ↑ T : la meilleure tournée de L_2.
Début
// BestFitness (↓ L_2 : une liste de tournées) est une fonction qui calcule la valeur de la fonction objectif de chaque tournée de L_2 et renvoie la tournée qui a la meilleure valeur.
T= BestFitness (L_2) ;
Fin

4.2.4.3 Sélection semi-élective

Le pseudocode du mode de selection semi-élective est présenté par l'algorithm 19, si le mode de sélection du prochain Roi se fait par sélection semi-élective, le choix du prochain Roi se fera à partir de la liste de dynasty $D = L_1 \cup L_2$.

Algorithm 19 Procedure Semi-élective (↓ L_1, L_2 : deux listes de tournées, ↑ T : une tournée)

Entrée ↓ L_1, L_2 : deux listes de tournées.
Sortie ↑ T : la meilleure tournée de $L_1 \cup L_2$.
Début
Tour D= $L_1 \cup L_2$; // BestFitness (↓ D : une liste de tournées) est une fonction qui calcule la valeur de la fonction objectif de chaque tournée de D et renvoie la meilleure tournée.
T= BestFitness (D) ;
Fin

4.2.4.4 L'intrus

Le pseudocode du mode de selection "Intrus" est présenté par l'algorithm 20, si le mode de sélection du prochain Roi est l'intrus, le prochain Roi sera construit en utilisant n'importe quelle métaheuristique connue. Dans notre cas, nous avons opté pour l'utilisation de l'algorithm du recuit simulé (SA).

Algorithm 20 Procedure Intrus (↑ $Solution$: une tournée)

Sortie ↑ Solution : une tournée trouvée en utilisant Recuit Simulé
Début
// FirstSolution() est une fonction qui trouve une première solution pour lancer l'algorithme du recuit simulé.
Tour CurrentTour=FirstSolution() ;
// Initialiser les paramètres du recuit simulés
Fixer les valeurs des températures $T_{Initial}$, T_{Final} ; Fixer I_{Max} ; T= $T_{initial}$;
while $i \leq I_{Max}$ **do**
  while $T \geq T_{Final}$ **do**
    // Choisir un voisin aléatoire de la tournée courante (CurrentTour).
    $NextTour \in N(CurrentTour)$;
    $\Delta E = Fitness(NextTour) - Fitness(CurrentTour)$;
    if $\Delta E \leq 0$ **then**
      CurrentTour= NextTour ;
    else if $\Delta E > 0$ **then**
      Mettre (CurrentTour=NextTour) avec probabilité $\exp^{-\frac{\Delta}{T}}$
    end if
    $T = T \times CoolingRate$;
  end while
  $i = i + 1$;
end while
Solution= CurrentTour ;
Fin

4.3 Résultats expérimentaux et discussion

La métaheuristique MN a été testée sur plusieurs instances de TSP euclidien et symetrique. Les instances ont été collectées à partir de TSPLIB qui est une bibliothèque d'instances de TSP disponible en ligne. La plupart des instances incluses dans TSPLIB ont déjà été résolues de manière optimale et ont été utilisées dans de nombreuses études de recherche. Chaque instance présente un nombre spécifique de villes (par exemple, les données de l'instance Eil51 incluent 51 villes et l'instance Pr76 comprend 76 villes, etc.).
La Monarchie Métaheuristique a été générée en 500 itérations et chaque problème de TSP a été répété 50 fois en utilisant des distances entières et 20 fois en utilisant des distances réelles. L'algorithme MN a été implémenté en utilisant Java sur un ordinateur Dell de RAM de 4 GB sous windows 7. Le paramètre de la métaheuristique MN correspond à la taille des deux listes L_1 et L_2. Pour la première itération, la taille initiale de L_1 et L_2 a été fixé à 50. Les résultats des autres méthodes ont été directement prise de la littérature. Les résultats présentés sont les meilleurs, les moyennes, les pires durées de tournées et les pourcentages d'erreurs relatives. Le pourcentage d'erreur relative a été calculé en utilisant l'équation suivante :

$$RE(\%) = \frac{(LRT - LOT)}{LOT} \times 100$$

LRT : durée de la tournée resultante, LOT : durée de la tournée optimale.

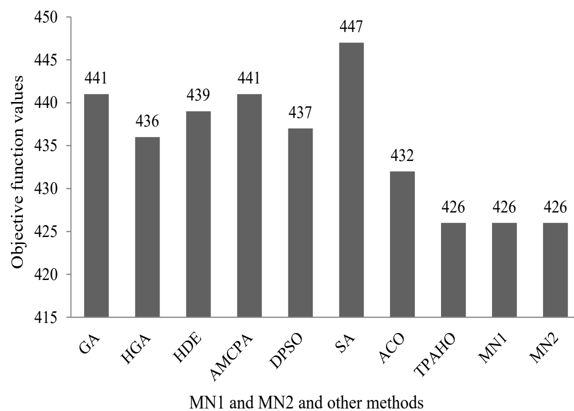


FIGURE 4.1 – Comparaison des valeurs de la fonction objectif pour l'instance Eil51 de TSP (distances entières).

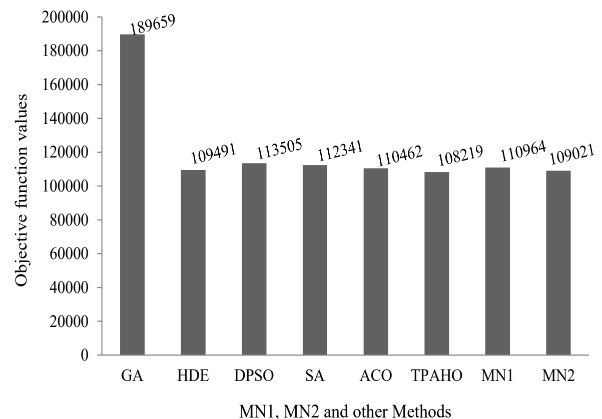


FIGURE 4.2 – Comparaison des valeurs de la fonction objectif pour l'instance Pr76 de TSP (distances entières).

Les Figures 4.1, 4.2 montrent une comparaison des valeurs de la fonction objectif de la

TABLE 4.1 – Comparaison de l'algorithme MN avec d'autres algorithmes existants dans la littérature (distances entières).

Problem	Method	Best	Average	Worst	RE	OPT
Eil51	GA	441	445	-	3.52	426
	HGA	436	440	-	2.35	
	AMCPA	441	465.7	-	3.52	
	HDE	439	444	-	3.05	
	DPSO	437	445	-	2.58	
	SA	447	467	-	4.92	
	ACO	432	439	-	1.40	
	TPASHO	426	431	-	0	
	MN_1	**426**	**432.88**	**442**	**0**	
	MN_2	**426**	**435.2**	**468**	**0**	
Berlin52	GA	7745	8040.1	-	2.69	7542
	AMCPA	7542	7805.2	-	0	
	HGA	7544	7559	-	0.03	
	HDE	7544	7816	-	0.02	
	DPSO	7700	7960	-	2.09	
	MN_1	**7863**	**8071.88**	**8183**	**4.25**	
	MN_2	**7703**	**7947.32**	**8295**	**2.13**	
St70	GA	707	750.2	-	4.74	675
	AMCPA	692	706.5	-	2.51	
	HGA	683	686	-	1.19	
	HDE	684	691	-	1.33	
	DPSO	712	733	-	5.48	
	MN_1	**705**	**721.32**	**735**	**4.44**	
	MN_2	**675**	**698.48**	**733**	**0**	

méthode proposée avec d'autres métaheuristiques existantes qui ont utilisé des distances entières pour les instances Eil51 et Pr76 de TSP.
Le tableau 4.1 montre une comparaison entre les deux variantes proposées de la monarchie métaheuristique MN_1, MN_2 et les autres méthodes de la littérature qui ont utilisé des distances entières, les résultats de GA (Genetic Algorithm) and AMCPA (Adaptive Multi-Crossover Population Algorithm) ont été pris de [119], les résultats de HGA (Hybrid Genetic Algorithm) ont été pris de [120], les résultats de HDE (Hybrid Differential Evolution Algorithm) et DPSO (Discrete Particle Swam Optimization) ont été pris de [121], les résultats de SA (Simulated Annealing), ACO (Ant Colony Optimization) and TPASHO (a Two Phase Hybrid Optimization Algorithm) ont été pris de [122].

TABLE 4.1 – (*Continued*).

Problem	Method	Best	Average	Worst	RE	OPT
	GA	558	610.6	-	3.71	
	AMCPA	566	578.1	-	5.20	
	HGA	552	559	-	2.60	
Eil76	HDE	558	568	-	3.71	538
	DPSO	580	587	-	7.80	
	MN_1	**573**	**580.4**	**589**	**6.50**	
	MN_2	**540**	**558.54**	**585**	**0.37**	
	GA	189659	-	-	75.35	
	HDE	109491	110539	-	1.23	
	DPSO	113505	115144	-	4.94	
Pr76	SA	112341	113523	-	3.86	108159
	ACO	110462	111037	-	2.13	
	TPASHO	108219	110031	-	0.05	
	MN_1	**110964**	**114522.66**	**116355**	**2.59**	
	MN_2	**109021**	**111116.08**	**115384**	**0.79**	
	GA	21566	22270.4	-	1.33	
Kroa100	AMCPA	21608	22125.3	-	1.53	
	HGA	21733	21811	-	2.12	21282
	MN_1	**23178**	**23625**	**23844**	**8.90**	
	MN_2	**22363**	**23398.08**	**23674**	**5.08**	
	GA	696	725.8	-	10.65	
Eil101	AMCPA	657	678.1	-	4.45	
	HGA	651	661	-	3.50	629
	MN_1	**654**	**667.38**	**674**	**3.97**	
	MN_2	**630**	**649.68**	**670**	**0.15**	

Comme le montre le Tableau 4.1, les résultats de la méthode proposée sont meilleurs que ceux de toutes les autres méthodes pour les instances $Eil101$, $Eil76$, $Eil51$, $St70$ et les valeurs optimales ont été atteints pour les instances $Eil51$ et $St70$. Pour l'instance $Pr76$, le resultat obtenu est supérieur à celui de GA, HDE, DPSO, SA, ACO et le resultat trouvé n'est pas loin de la valeur optimale de la fonction objectif ($RE = 0.79$).

Nous avons remarqué que les résultats de MN_2 sont meilleurs que ceux de MN_1, raison pour laquelle nous avons effectué d'autres tests en utilisant des distances réelles pour pouvoir comparer MN_2 avec des méthodes de la littérature qui utilisent des distances réelles.

Le Tableau 4.2 montre une comparaison entre la méthode proposée MN_2 et celles d'ACO

(optimisation de la colonie de fourmis), ABC (colonie d'abeilles artificielles) et HA (approche hiérarchique : ACO avec ABC), ces trois méthodes ont utilisé des distances réelles et leurs résultats ont été pris de l'article [123].

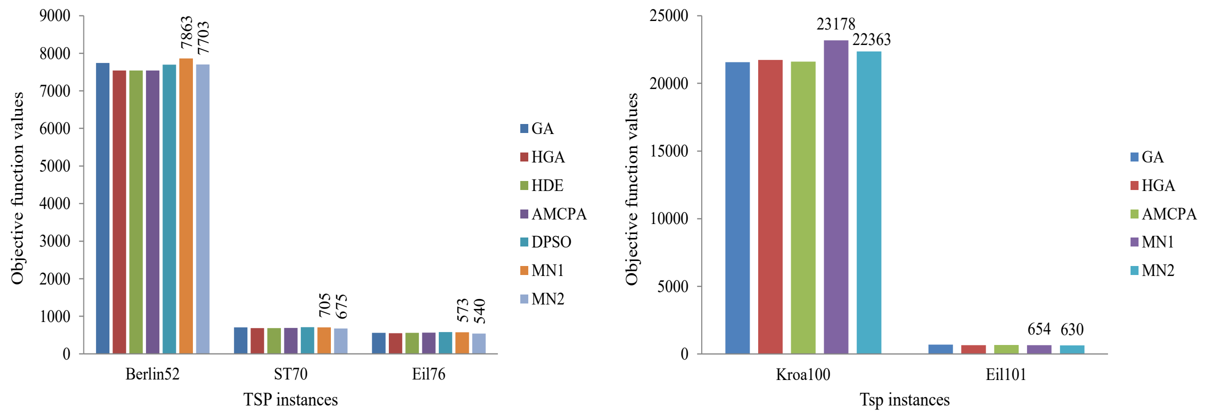


FIGURE 4.3 – Comparaison des valeurs de la fonction objectif pour Berlin52, Eil76, St70 instances de TSP (distances entières).

FIGURE 4.4 – Comparaison des valeurs de la fonction objectif pour Kroa100, Eil101 instances de TSP (distances entières).

A partir du Tableau 4.2, nous pouvons voir que la méthode proposée (MN) a produit des résultats meilleurs que ACO en termes de temps et de qualité. Les resultats obtenus par la méthode MN sont meilleurs que ceux de la méthode ABC, la méthode MN est aussi très compétitive avec la méthode hybride HA.

Les Figures 4.3, 4.4 montrent une comparaison des valeurs de la fonction objectif de la méthode proposée avec d'autres métaheuristiques existantes qui ont utilisé des distances entières.

La Figure 4.5 montre une comparaison des pourcentages d'erreurs relatives de notre métaheuristique avec ceux de d'autres métaheuristiques existantes dans la littérature qui ont utilisé des distances réelles.

La Figure 4.6 montre une comparaison des pourcentages d'erreurs relatives de notre métaheuristique avec ceux de d'autres métaheuristiques existantes dans la littérature qui ont utilisés des distances entières.

Les Figures 4.7, 4.8 présentent une comparaison des valeurs de la fonction objectif de la méthode proposée avec d'autres métaheuristiques existantes dans la littérature qui ont utilisé des distances réelles. La Figure 4.9 montre la comparaison du temps de calcul entre la métaheuristique MN et les autres méthodes existantes dans la littérature.

TABLE 4.2 – Comparaison de l'algorithme MN avec d'autres algorithmes existants dans la littérature (distances réelles).

Problem	Method	Best	Average	Worst	RE	OPT	CPU(s)
Eil51	ACO	450.59	457.86	463.55	5.06	428.87	112.11
	ABC	563.75	590.49	619.44	31.45		2.16
	HA	431.74	443.39	454.97	0.67		58.33
	MN_2	**431.17**	**437.25**	**446.89**	**0.54**		**71.1**
Berlin52	ACO	7548.99	7659.31	7681.75	0.06	7544.37	116.67
	ABC	9479.11	10390.26	11021.99	25.64		2.17
	HA	7544.37	7544.37	7544.37	0		60.64
	MN_2	**7774.24**	**7952.09**	**8065.97**	**3.05**		**69.25**
St70	ACO	696.05	709.16	725.26	2.79	677.11	226.06
	ABC	1162.12	1230.49	1339.24	71.63		3.15
	HA	687.24	700.58	716.52	1.49		115.65
	MN_2	**682.66**	**706.11**	**724.59**	**0.82**		**103.8**
Eil76	ACO	554.46	561.98	568.62	1.66	545.39	271.98
	ABC	877.28	931.44	971.36	60.85		3.49
	HA	551.07	557.98	565.51	1.04		138.82
	MN_2	**563.66**	**573.68**	**583.43**	**3.35**		**123.85**
Pr76	ACO	115166.66	116321.22	118227.41	6.48	108159.44	272.41
	ABC	195198.9	205119.61	219173.64	80.47		3.50
	HA	113798.56	115072.29	116353.01	5.21		138.92
	MN_2	**109140.82**	**111049.24**	**114868.83**	**0.91**		**125.9**
Kroa100	ACO	22455.89	22880.12	23365.46	5.49	21285.44	615.06
	ABC	49519.51	53840.03	57566.05	132.64		5.17
	HA	22122.75	22435.31	23050.81	0.04		311.12
	MN_2	**22415.47**	**23116.88**	**23703.12**	**5.31**		**204.65**
Eil101	ACO	678.04	693.42	705.65	5.56	642.31	527.42
	ABC	1237.31	1315.95	1392.64	92.63		5.17
	HA	672.71	683.39	696.04	4.73		267.08
	MN_2	**664.258**	**682.935**	**700.497**	**3.42**		**210.05**

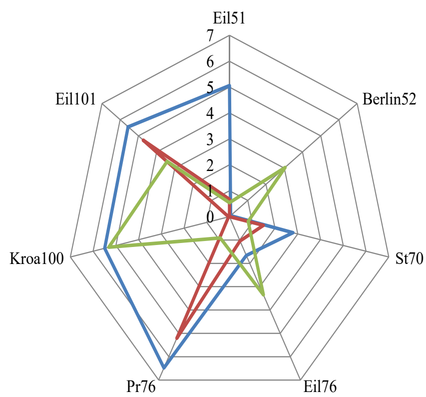


FIGURE 4.5 – Comparaison des RE% de MN2 avec ceux de ACO et HA (distances réelles).

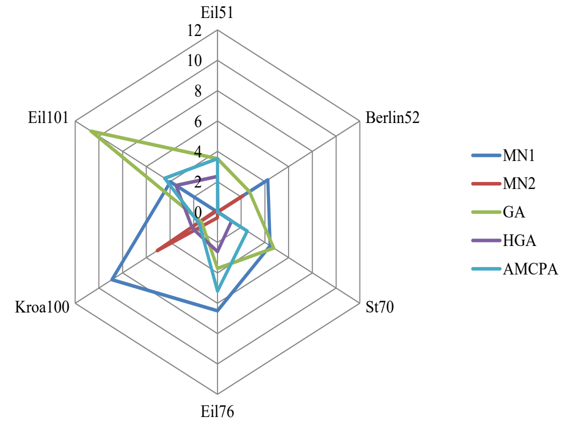


FIGURE 4.6 – Comparaison des RE% de MN1, MN2 avec ceux de ACO, ABC, HA (distances entières).

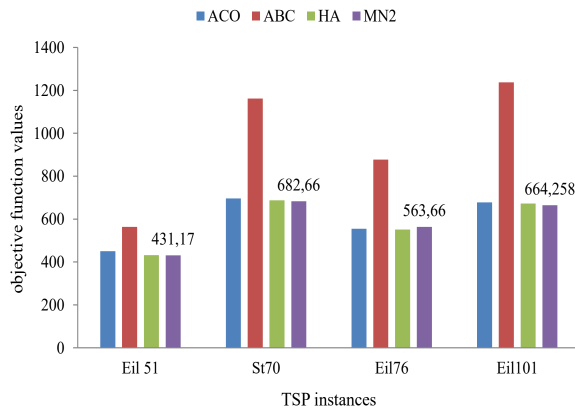


FIGURE 4.7 – Comparaison des valeurs de la fonction objectif pour les instances Eil51, St70, Eil76, Eil101 de TSP (distances réelles).

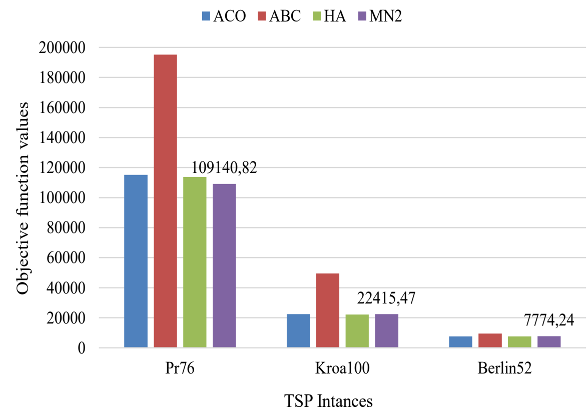


FIGURE 4.8 – Comparaison des valeurs de la fonction objectif pour les instances Pr76, Kroa100, Berlin52 de TSP (distances réelles).

La Figure 4.10 montre la courbe de convergence des meilleurs résultats pour la méthode proposée (la deuxième alternative MN_2) en une seule exécution (meilleure exécution) pour l'instance Eil51 de TSP. Dans ce cas là, L'algorithme a démarré avec une solution trouvée par une heuristique gloutonne et a atteint la valeur 426, qui est la valeur optimale pour l'instance Eil51.
La Figure 4.11 montre la courbe de convergence des meilleurs résultats pour la méthode proposée (la seconde alternative MN_2) en une seule exécution pour Pr76, l'algorithme proposé a démarré avec une solution trouvée par une heuristique gloutonne.

Les Figures 4.12a, 4.12b montrent les courbes de convergence de la méthode proposée

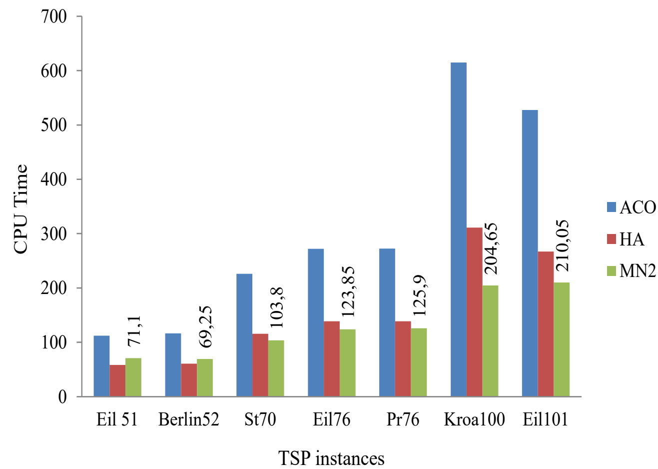


FIGURE 4.9 – Comparaison des temps de calcul de ACO, HA (ACO+ABC) avec celui de MN2 (distances réelles).

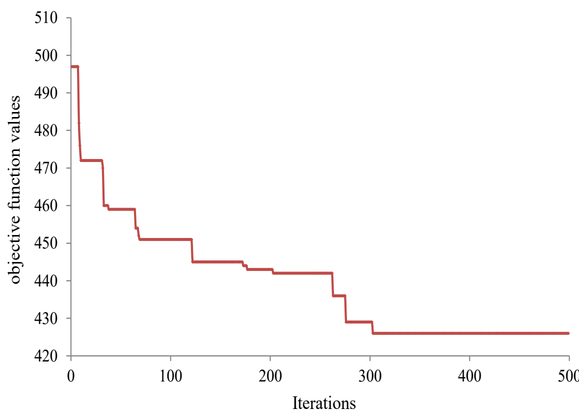


FIGURE 4.10 – Courbe de convergence des meilleurs résultats pour MN_2 pour l'instance Eil51 en démarrant par une solution trouvée par une heuristique gloutonne.

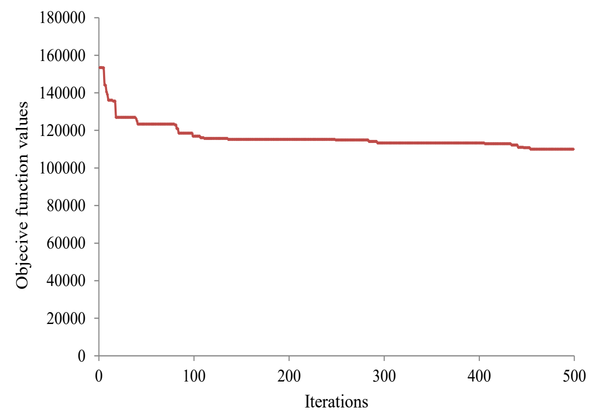
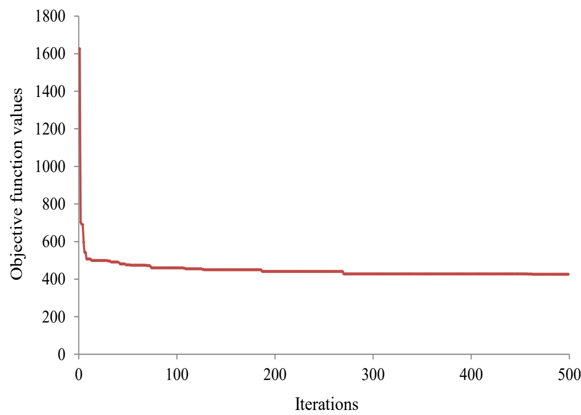


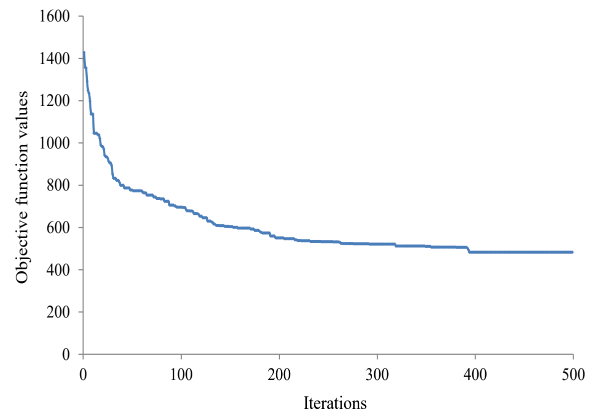
FIGURE 4.11 – Courbe de convergence des meilleurs résultats pour MN_2 pour l'instance Pr76 en démarrant par une solution trouvée par une heuristique gloutonne.

et de l'algorithme génétique pour l'instance Eil51. Les Figures 4.13a, 4.13b montrent les courbes de convergence de la méthode proposée et l'algorithme génétique pour l'instance Pr76.

Les Figures 4.12, 4.13 montrent une comparaison entre la métaheuristique proposée et l'algorithme génétique des courbes de convergence des meilleurs résultats obtenus, les deux méthodes ont démarré avec des solutions générées aléatoirement pour les instances Eil51 et Pr76. L'algorithme proposé converge plus rapidement que l'algorithme génétique et nécessite moins d'itérations que l'algorithme génétique pour atteindre l'optimum global.

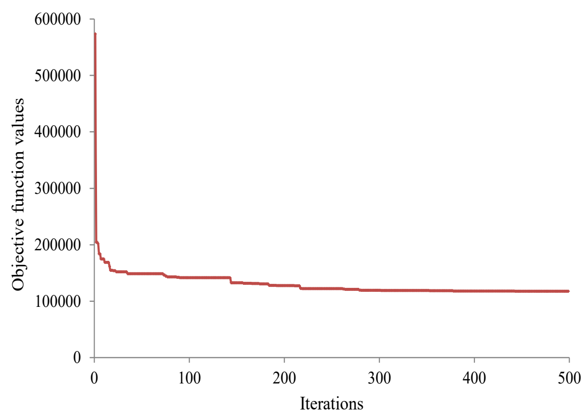


(a) Courbe de convergence des meilleurs résultats pour MN_2

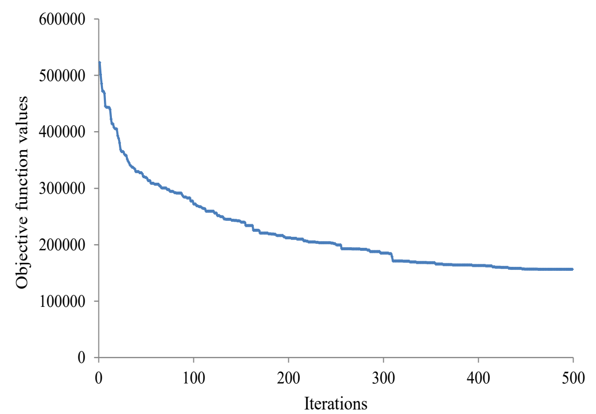


(b) Courbe de convergence des meilleurs résultats pour GA

FIGURE 4.12 – Comparaison des courbes de convergence entre MN_2 et GA, les deux ont démarré par des solutions générées aléatoirement pour l'instance Eil51.



(a) Courbe de convergence des meilleurs résultats pour MN_2



(b) Courbe de convergence des meilleurs résultats pour GA

FIGURE 4.13 – Comparaison des courbes de convergence entre MN_2 et GA, les deux ont démarré par des solutions générées aléatoirement pour l'instance Pr76.

Comme le montrent les Figures 4.12, 4.13, la méthode proposée (métaheuristique MN) surpasse l'algorithme génétique en termes de convergence et de qualité de solution.

Les résultats obtenus pour les instances TSP prouvent que la méthode proposée (la Monarchie Métaheuristique) est compétitive avec les méthodes existantes dans la littérature.

Il est à noter que la métaheuristique MN a également la possibilité d'améliorer ses performances car elle peut utiliser divers algorithmes comme sous-algorithmes.
Dans le chapitre qui va suivre nous expliquerons comment adapter la monarchie métaheuristique à la résolution de d'autres problèmes d'optimisation combinatoire tels que le problème CVRP et le problème du sac à dos.

Don't let the noise of others'opinions drown out your own inner voice. And most important, have the courage to follow your heart and intuition.

Steve Jobs

CHAPITRE 5

ADAPTATION DE LA MONARCHIE METAHEURISTIQUE (MN) À LA RÉSOLUTION DE D'AUTRES PROBLÈMES D'OC

Pour prouver la simplicité d'adaptation de la monarchie métaheuristique nous allons expliquer dans ce chapitre comment peut on l'adapter à d'autres problèmes d'optimisation combinatoire, nous illustrons l'application sur le problème de le problème de sac à dos binaire et le problème de tournées de véhicules.

5.1 Adaptation de MN au problème du sac à dos binaire

Le problème de sac à dos en $0-1$ (ou binary knapsack problem, noté 0-1 KP) est l'un des plus connus des problèmes d'optimisation combinatoire. Rappelons que pour le problème de sac à dos on a un ensemble d'objets de 1 à n. Chaque objet $j \in 1, \ldots, n$ a un poids w_j de valeur entière et rapporte un profit p_j . On dispose d'un sac dont le contenu ne doit pas excéder une capacité c de valeur entière fixée et on désire le remplir de façon à maximiser la somme des profits relatifs aux objets placés, et ce en respectant la contrainte de capacité. Les solutions de ce problème peuvent être le mieux codé avec une représentation binaire, chaque bit représente la présence ou l'absence d'un élément dans le sac à dos. Par exemple la représentation 1100111 indique que l'element $1, 2, 5, 6, 7$ existent dans le sac à dos et $3, 4$ n'existent pas.

5.1.1 Création de la première solution Roi

L'heuristique gloutonne suivante peut être utilisée pour construire la première solution Roi pour le sac à dos binaire :

Algorithm 21 **Création** $1^{ère}$ **solution Roi pour 0-1 KP**

Input : Instance de sac à dos binaire (On a un sac de capacité c, n objets, chaque objet j a un poids w_j et un profit p_j).
Output : Une solution réalisable $\overline{x}$.
Trier par ordre décroissant les objets en fonction de leur rapport du profit sur le poids (c-à-d $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$).
Sélectionner de façon gloutonne un article selon l'ordre défini précédemment.
$\overline{c}$ représente la capacité restante dans le sac à dos, $\overline{c} \leftarrow c$;
for $j = 1$ à n **do** faire
  if $w_j \leq \overline{c}$ **then**
    $\overline{x_j} = 1$;
    $\overline{c} = \overline{c} - w_j$;
  else if $w_j > \overline{c}$ **then**
    $\overline{x_j} = 0$;
  end if
end for
Return $\overline{x}$;
End.

5.2 Création de la liste L_1

La liste L_1 dans le cas du KP binaire, est crée en utilisant les operateurs adéquats à la représentation binaire adopté pour ce problème. En plus des operateurs de croisement à un point de croisement, à deux point de croisement, à k points de croisement et le croisement uniforme qui peuvent être adapter à la représentation binaire aussi (illustré par les figurent respectivement), on a des operateurs dédiés spécialement à la représentation binaire qu'on appele opérateurs de croisement binaire (binary crossover), nous présentons quelques-uns dans ce qui suit :

5.2.1 Le croisement uniforme [5]

Cet operateur génére les vecteurs enfants bit par bit à partir des deux parents, un masque est utilisé, s'il est égal à 1, l'enfant 1 reçoit l'élément correspondant du parent 1 et l'enfant 2 reçoit celui du parent 2. Sinon, l'échange se fait dans l'autre sens.

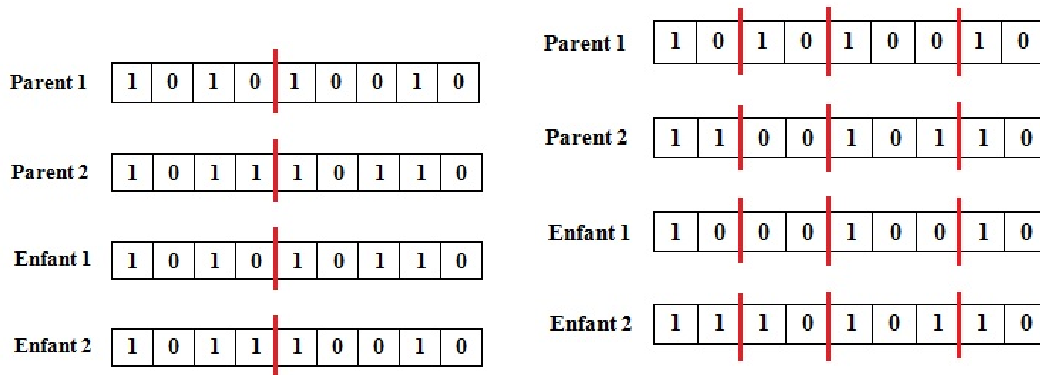


FIGURE 5.1 – Exemple d'application de croisement à un point

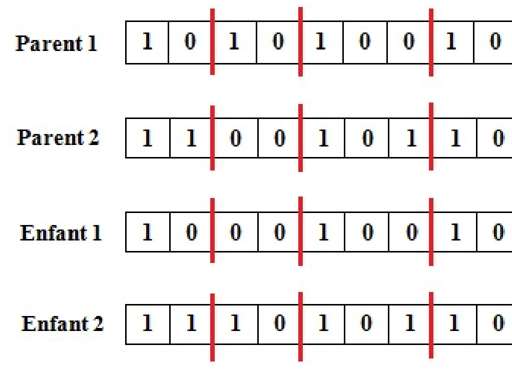


FIGURE 5.2 – Exemple d'application de croisement à k points (pour cet exemple on a 4 points de croisement)

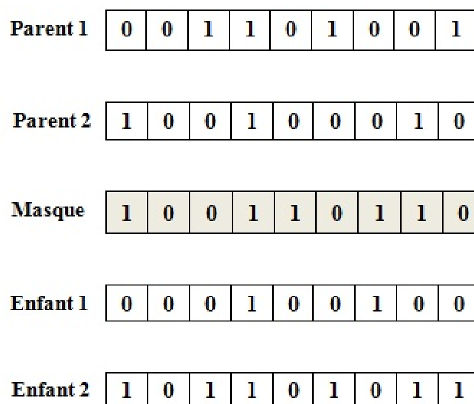


FIGURE 5.3 – Exemple d'application de croisement uniforme

5.2.2 le croisement 1-BX [5]

Dans la méthode 1- Bit adaptation (1-BX), le dernier bit du vecteur solution est réservé au code de l'un des deux operateurs de croisement appliqués. En supposant que 0 correspond au croisement uniforme (UX) et 1 correspond au croisement à 2 points (2-PX), le choix de l'un d'entre eux se fait en fonction de la règle : si le dernier bit des parents est le même alors choisissez l'opérateur indiqué par ce bit. Sinon choisir un opérateur en sélectionnant par tirage au sort, c'est-à-dire en sélectionnant l'uniforme nombre réel aléatoire de 0 à 1. Si cette valeur est $< (0,5)$, alors Le croisement uniforme est effectué sinon le croisement à 2 points est utilisé.

5.2.3 Le croisement RRC [6] [5]

Pour le croisement Random Respectful (RRC), on sélectionne deux parents A et B pour le croisement et l'enfant est généré en se basant sur le vecteur de similarité des parents. Le vecteur de similarité est crée comme suit : $S_{AB} = (s_1^{AB}, s_2^{AB}, \ldots, s_n^{AB})$ tel que si pour une position donnée les deux parents ont les mêmes valeurs alors on affecte la même valeure à la même position dans vecteur similarité dans le cas contraire on affecte nulle à cette position dans le vecteur similarité. Après création du vecteur similarité S_{AB}, deux vecteurs enfants seront crées selon les valeures du vecteur similarité. Si le vecteur similarité contient 1 alors les deux vecteurs enfants contiendront 1 à la même position, si le vecteur similarité contient 0 alors les deux vecteurs enfants contiendront 1 à la même position, s'il contient $null$ alors la valeure affectée au vecteur enfant est séléctionnée en prenant une valeure aléatoire réelle dans l'intervalle $[0, 1]$, si cette valeure est < 0.5 alors on enregistre 1 sinon 0 est enregistré, on procede de la même manière pour compléter le deuxième vecteur enfant.

Plusieurs autres types de croisement existent dans la litterature et peuvent être utilisés tels que : circle ring crossover [124], diagonal crossover [125], best combinatorial crossover(BCX) [126] etc.

5.3 Creation de la liste L_2

Pour remplir la liste L_2 on peut utiliser le voisinage suivant pour créer l'ensemple des voisins de la solution Roi.

5.3.1 Voisinage par complémentation (remplacement) :

Pour les solutions codées sous forme d'une chaîne en $0-1$. la transformation suivante peut être utilisée : on remplace un $0-1$ quelconque de la chaîne par son complémentaire.
Par exemple : 100111001 devient 100110001 par complémentation du sixième élément, dans le cas du problème de sur sac à dos ceci revient à choisir un objet j, et à changer la décision le concernant : $x_j \leftarrow 1 - x_j$. La solution doit être admissible pour être dans le voisinage). On peut généraliser cette transformation (voisinage de taille k), on choisit k objets, et on change la décision les concernant, en vérifiant l'admissibilité. Un voisinage de taille k consiste à modifier k variables. Il y a autant de voisins que de combinaisons de k objets parmi n (c-à-d $\frac{n!}{k!(n-k)!}$).

5.4 Stratégies utilisées dans le cas d'irréalisabilité d'une solution

Après avoir appliquer les operateurs de coisement ou même dans le cas de réalisation du voisinage de remplacement, on peut avoir des solutions qui ne sont pas valides pour le problème de sac à dos binaire, pour remédier à ce problème on a peut utiliser une des deux stratégies suivantes :

1. **Algorithmes basés sur des méthodes de pénalité**
   On affecte à une solution valide une pénalité 0 et à une solution non valide une pénalité ≥ 0.

2. **Algorithme basés sur des méthodes de réparation** On peut faire par exemple :

   - **Une réparation aléatoire (random repair)** : L'élément à éliminer du sac à dos est séléctionné alétoirement.
   - **Réparation gloutonne (Greedy repair)** : On ordonne tous les elements du sac à dos par ordre décroissant selon leur rapport bénéfice/poids, l'element à éliminer de la solution est le dernier.

5.5 Qu'elle méthode de résolution utiliser pour créer la prochaine solution Roi quand le mode de selection du prochain Roi est l'intrus :

On peut utiliser n'importe quelle métaheuristique existante déjà pour la résolution du problème de sac à dos binaire, parmis ces méthodes on peut citer : Le recuit simulé [127], les algorithme génétique (GA) [128], les colonies de fourmis [129], quantum-inspired evolutionary algorithm [130], schema-guiding evolutionary algorithm [131], global harmony search algorithm [132], etc.

5.6 Adaptation de MN au problème VRP avec contrainte de capacité

La version la plus élémentaire du VRP est le problème de tournées de véhicules avec contraintes de capacité (CVRP). Le CVRP présente un certain nombre de caractéristiques et de contraintes

spécifiques. Il n'y a qu'un seul dépôt de véhicules. Chaque véhicule a une capacité limitée Q disponible pour livrer des marchandises. Chaque véhicule a également une durée de temps limitée de service (conduite) chaque jour. Il existe un certain nombre de clients et chaque client visité par un véhicule a une certaine demande qui occupe une partie de la capacité de ce véhicule. Il n'est pas possible de fractionner la livraison des marchandises destinées à un client.
Une solution CVRP est donc un ensemble de tournées avec une distance parcourue totale minimale. Ces tournées doivent rendre visite à chaque client exactement une seule fois, la demande totale de chaque tournée est au maximum Q. La capacité et la durée de conduite maximale sont identiques pour chaque véhicule. Tous les véhicules sont donc du même type. Lorsque tous les véhicules d'une flotte sont identiques, on parle de flotte homogène.
Pour le CVRP, un certain nombre d'algorithmes exacts tels que Branch and Bound, Branch and Cut et la programmation dynamique sont connus pour résoudre ce problème. Cependant, ces algorithmes necessite un temps de calcul impotant. Ceci n'est pas d'une grande utilité pratique, donc ces méthodes ne sont généralement pas utilisées dans la pratique. C'est pourquoi les méthodes approchées sont faforisées face au problèmes pratiques de grande taille. Étant donné que le CVRP la version basique du VRP, il est généralement appelé VRP.

5.6.1 Création de la première solution Roi

Le premier Roi peut être crée dans le cas du CVRP par des méthodes constructives, dans lesquelles les solutions sont construites progressivement. Les principales caractéristiques des heuristiques constructives sont la simplicité et la rapidité à trouver des solutions bonnes et acceptables. Les heuristiques les plus connues pour CVRP sont la méthode de Clarke et Wright [133] qui consiste à affecter un véhicule à chaque client pour ensuite fusionner les tournées de manière à réduire le coût total, un tour géant est obtenu en concaténant les tournées. On peut aussi utiliser l'algorithme de balayage [134], l'heuristique d'insertion [135], l'algorithme de pétale ou l'algorithme [136] du plus proche voisin.
Dans ce qui suit nous présentons l'algorithme 22 qui représente un algorithme heuristique qui peut être utiliser pour la création de la première solution Roi. Les solutions sont codés comme des séquences de n clients sans délimiteurs de tournées. On obtient ainsi un tour géant qui sera découpé en tournées réalisables à l'aide d'une procédure de découpage utilisant l'opérateur Split de (Prins 2004) [137].

Algorithm 22 Heuristique de construction de la première solution Roi (cas : CVRP)

Input C : set of all cities to be visited.
Output V : a CVRP solution.
Begin
Construire une tournée géante.
diviser la tournée géante en sous-tournées selon les demandes des clients et la capacité du vehicule
Améliorer chaque sous-tournée par l'algorithme du plus proche voisin.
End

5.6.2 Création de la liste L_1

Dans la litérature, dix types d'opérateurs de croisement ont été déjà appliqués pour le CVRP : One-Point Crossover (1PX), Two-Point Crossover (2PX), Order Crossover (OX), Partially Mapped Crossover (PMX), Cyclical Crossover (CX), Route Based Crossover (RBX) [138, 139], Sequence-Based Crossover (SBX) [140], Single Parent Crossover (SPO) [141], Genetic Vehicle Representation Crossover (GVR) [142] and Best Cost-Best Route Crossover (BCBRC) [143, 144]
Pour la création de la liste L_1, on peut choisir parmis les opérateurs cités ci dessus, l'essentiel c'est de vérifier que la solution qui résulte par application de l'opérateur choisit soit réalisable sinon des procedures de correction de solution doivent être utilisés pour garantir la validité de la solution. Il est à noter aussi que l'opérateur de croisement utilisé dépend de la représentation de donnée adoptée.

5.6.3 Création de la liste L_2

Cette liste sera remplie par des solutions voisines de la solution Roi courante. Nous citons sept opérateurs de voisinage qui peuvent être utilisés. Ils peuvent être classés en deux classes : les opérateurs intra-route qui opérent sur une même tournée, et les opérateurs inter-routes qui opèrent entre deux tournées.

5.6.3.1 Les voisinages qui opérent sur une même tournée (intra route operators :

On retrouve dans cette classe le voisinage à un point (move), à deux points (swap), le voisinage 2-opt,.. etc.

5.6.3.2 Les voisinages qui opérent sur deux tournées) (inter route operators :

On peut citer le voisinage 1-0 échange (1-0 exchange neighborhood) où un client est supprimer d'une tournée donnée et est inseré dans la meilleure position à la deuxième tournée. On a aussi Le voisinage 1-1 échange (1-1 exchange operator) où un client est supprimée de chaqu'une des deux tournée et est inseré dans la meilleure position dans l'autre tournée.

5.6.4 Qu'elle méthode de résolution utiliser pour créer la prochaine solution Roi quand le mode de selection du prochain est l'intrus :

Dans le cas du CVRP nous proposons l'utilisation de la métaheuristique de recherche tabou pour création du prochain Roi quand la séléction du prochain Roi est par l'intrus. La recherche tabou est parmis les métaheuristiques les plus performantes pour la résolution du CVRP d'après des études faites par des chercheurs. Plusieurs variantes de tabou existent dans la literature qui peuvent être adapter à CVRP on peut citer : Osman's algorithm [145], the Gendreau, Hertz and Laporte "Taburoute algorithm" [146], Taillard's algorithm [147], the Rochat and Taillard Adaptive Memory Procedure [148], the Xu and Kelly algorithm [149], the Rego and Roucairol Tabuchain algorithm [150], Rego's Flower algorithm [151], the Toth and Vigo granular tabu search algorithm [152], the Unified Tabu Search Algorithm of "Cordeau, Gendreau, Laporte and Mercier" [153].

La simplicité d'adaptation de la monarchie métaheuristique est l'un des points forts de la méthode. En effet, de manière générale, il suffit de choisir une bonne représentation de solutions qui nous permet de choisir des opérateurs de croisement (pour former les élements de la liste L_1) et des oprérateurs de voisinages (qui vont permettre de remplir la liste L_2) qui fournissent des solutions valides au problème traité sinon on aura à appliquer des procedures de correction de solutions aussi. Une étape importante de la monarchie métaheuristique aussi est de choisir une bonne méthode pour créer la solution intrus et là l'utilisateur à l'ambarras du choix, plusieurs démarches peuvent être envisager : On peux chercher la meilleure métaheuristique qui existe dans la literature pour le problème à traiter, ou bien appliquer une nouvelle heuristique relative au problème traité.

I do not think there is any thrill that can go through the human heart like that felt by the inventor as he sees some creation of the brain unfolding to success . . . Such emotions make a man forget food, sleep, friends, love, everything.

Nikola Tesla

CONCLUSION ET PERSPECTIVES

Les métaheuristiques sont des moyens efficaces pour résoudre les problèmes d'optimisation combinatoire dans les plus cours délais. En effet, la complexité non polynomiale de la majorité des problèmes de cette classe et leur importance liée à leur forte présence dans les secteurs academiques et économiques poussent les chercheurs à developer de manière continue de nouvelles méthodes notamment des méthodes approchées dans le but de garantir de trouver de bonnes solutions en un laps de temps raisonable.

La première partie de cette thèse a présenté le contexte et l'état de l'art de notre domaine de recherche, un premier chapitre introductif sur l'optimisation combinatoire suivi par un deuxième chapitre où les différentes connaissances de base relatives au méthodes de résolution approchées ont été présentées.

Dans la deuxième partie de cette thèse les contributions réalisées ont été présentés. En effet, durant ce travail doctoral, une une nouvelle métaheuristique intitulée la monarchie métaheuristique (MN) a été developée, cette méthode a été présentée dans le chapitre trois. Le chapitre quatre, comporte une explication détaillée et les resultats d'une première variante de cette métaheuristique dédiée à la résolution d'un problème très connu en recherche opérationnelle qui est le problème de voyageur de commerce connu par son acronyme anglophone (TSP) (Traveling Salesman Problem).

Les performances de la méthode developée MN ont été validées par le biais d'un ensemble de sept instances de TSP disponibles dans la bibliothèque d'instances en ligne TSPLIB. Les tests ont été effectués en utilisant des distances entières et des distances réelles. Les résultats de la méthode proposée indiquent que la méthode MN est compétitive par rap-

port aux autres méthodes existantes dans la littérature.

L'un des points forts de la méthode est la facilité de son adaptation, le dernier chapitre a été consacré à expliquer la manière d'adaptation de la méthode à la résolution d'un autre type de problème d'optimisation combinatoire qui est le problème du sac à dos. Une adaptation de la méthode à la résolution du problème qui représente une extension du problème TSP qui est le problème de tournées de véhicules (VRP) a été présenté dans le même chapitre aussi.

Selon le chercheur Silver 1980 [154], une bonne méthode approchée doit avoir les trois propriétés suivantes : La solution fournie par la méthode doit être proche de l'optimum, la probabilité de tomber sur une mauvaise solution doit être faible, et la méthode doit être aussi simple que possible pour que l'utilisateur puisse la comprendre, et la monarchie métaheuristique regroupe les trois propriétés, ce qui fait d'elle une bonne méthode approchée.

Cette thèse nous permet d'ouvrir plusieurs perspectives. En ce qui concerne les performances de la méthode sur d'autres problèmes, une implémentation ainsi qu'une analyse expérimentale de la méthode dans le cas du sac à dos et VRP seront faites comme suite directe de ce qui a été présenté en chapitre cinq. Aussi, d'autres variantes seront developée pour la résolution de d'autres type de problèmes d'optimisation combinatoire.

BIBLIOGRAPHIE

[1] Christian Blum and Günther R Raidl. *Hybrid Metaheuristics : Powerful Tools for Optimization*. Springer, 2016.

[2] HAJ-RACHID Mais, Christelle Bloch, Wahiba Ramdane-Cherif, and Pascal Chatonnay. Différentes opérateurs évolutionnaires de permutation : sélections, croisements et mutations. *Laboratoire d'informatique De L'université de franche-compte*, 2010.

[3] SN Sivanandam and SN Deepa. Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37.

[4] Leticia Hernando Rodríguez. *Instances of combinatorial optimization problems : complexity and generation*. PhD thesis, Universidad del País Vasco-Euskal Herriko Unibertsitatea, 2015.

[5] AJ Umbarkar and PD Sheth. Crossover operators in genetic algorithms : A review. *ICTACT journal on soft computing*, 6(1), 2015.

[6] Tomasz Dominik Gwiazda. *Crossover for single-objective numerical optimization problems*, volume 1. Tomasz Gwiazda, 2006.

[7] Sunith BANDARUA and Kalyanmoy DEBB. Metaheuristic techniques. *Decision Sciences : Theory and Practice*, 220(4598) :693–750, 2016.

[8] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11) :1097–1100, 1997.

[9] Jean-Charles Boisson. *Modélisation et résolution par métaheuristiques coopératives : de l'atome à la séquence protéique*. PhD thesis, Université Lille 1, 2008.

[10] Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. Metaheuristic research : a comprehensive survey. *Artificial Intelligence Review*, pages 1–43, 2018.

[11] Laetitia Jourdan. *Métaheuristiques Coopératives : du déterministe au stochastique*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2010.

[12] C.S. Beightler, D.T. Phillips, and D.J. Wilde. *Foundations of Optimization*. International Industrial and Systems Engineering Series. Prentice-Hall, 1979.

[13] Ibtissam Ahmia and Meziane Aider. A novel metaheuristic optimization algorithm : the monarchy metaheuristic. *Turkish Journal of Electrical Engineering & Computer Sciences*, 27(1) :362–376, 2019.

[14] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[15] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[16] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2) :83–97, 1955.

[17] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1) :80–91, 1959.

[18] Samuel Raff. Routing and scheduling of vehicles and crews : The state of the art. *Computers & Operations Research*, 10(2) :63–211, 1983.

[19] Nicos Christofides. The vehicle routing problem. *Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle*, 10(V1) :55–70, 1976.

[20] Gilbert Laporte. *Développements algorithmiques récents et perspectives de recherche en distributique*. Université de Montréal, Centre de recherche sur les transports, 1989.

[21] Martin Desrochers, Jan Karel Lenstra, and Martin WP Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46(3) :322–332, 1990.

[22] Paolo Toth and Daniele Vigo. An overview of vehicle routing problems. In *The vehicle routing problem*, pages 1–26. SIAM, 2002.

[23] Nicos Christofides. Vehicle routing. In *The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization*, pages 431–448. John Wiley and Sons, 1985.

[24] Lawerence Bodin, Bruce Golden, Arjang Assad, and Mark Ball. The state of the art in the routing and scheduling of vehicles and crews. *Computers and Operations Research*, 10(1) :63–211, 1983.

[25] Minea Filipec, Davor Skrlec, and Slavko Krajcar. An efficient implementation of genetic algorithms for constrained vehicle routing problem. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 3, pages 2231–2236. IEEE, 1998.

[26] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 337–347. IEEE, 1986.

[27] Daniel Pierre Bovet, Pierluigi Crescenzi, and D Bovet. *Introduction to the Theory of Complexity*. Citeseer, 1994.

[28] T Stützle. Local search algorithms for combinatorial problems. *Darmstadt University of Technology PhD Thesis*, 20, 1998.

[29] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[30] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[31] Bellman Richard. Notes on the theory of dynamic programming–transportation models. *Management Science*, 4 :191–195, 1958.

[32] Gerhard J Woeginger. Exact algorithms for np-hard problems : A survey. In *Combinatorial Optimization-Eureka, You Shrink !*, pages 185–207. Springer, 2003.

[33] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica : Journal of the Econometric Society*, pages 497–520, 1960.

[34] John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6) :972–989, 1963.

[35] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization : algorithms and complexity*. Courier Corporation, 1998.

[36] Ralph E Gomory. An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64 :260–302, 1963.

[37] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, pages 65–77, 2002.

[38] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3) :487–512, 2002.

[39] Claudio Contardo, Jean-François Cordeau, and Bernard Gendron. A computational comparison of flow formulations for the capacitated location-routing problem. *Discrete Optimization*, 10(4) :263–295, 2013.

[40] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price : Column generation for solving huge integer programs. *Operations research*, 46(3) :316–329, 1998.

[41] Martin Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations research*, 45(6) :831–841, 1997.

[42] George L Nemhauser, AHG Rinnooy Kan, and Michael J Todd. Optimization, vol. 1. *Handbooks in operations research and management science*, 1989.

[43] Gabriel Gutiérrez-Jarpa, Guy Desaulniers, Gilbert Laporte, and Vladimir Marianov. A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research*, 206(2) :341–349, 2010.

[44] Christian Blum, Jakob Puchinger, GR Raidl, Andrea Roli, et al. A brief survey on hybrid metaheuristics. *Proceedings of BIOMA*, pages 3–18, 2010.

[45] Colin R Reeves. *Modern heuristic techniques for combinatorial problems. Advanced topics in computer science*. Mc Graw-Hill, 1995.

[46] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[47] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549, 1986.

[48] Colin R Reeves. *Modern heuristic techniques for combinatorial problems. Advanced topics in computer science*. Mc Graw-Hill, 1995.

[49] Ibrahim H Osman and Gilbert Laporte. Metaheuristics : A bibliography, 1996.

[50] Kenneth Sörensen and Fred W Glover. Metaheuristics. In *Encyclopedia of operations research and management science*, pages 960–970. Springer, 2013.

[51] John Holland. Adaptation in natural and artificial systems : an introductory analysis with application to biology. *Control and artificial intelligence*, 1975.

[52] Marco Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[53] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[54] Marco Dorigo and Thomas Stützle. Ant colony optimization : overview and recent advances. In *Handbook of metaheuristics*, pages 311–351. Springer, 2019.

[55] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[56] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[57] Thomas A Feo and Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2) :67–71, 1989.

[58] Thomas A FEO and Mauricio GC RESENDE. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2) :109–133, 1995.

[59] Nenad Mladenovic. A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. In *papers presented at Optimization Days*, volume 12, 1995.

[60] Nils Aall Barricelli et al. Esempi numerici di processi di evoluzione. *Methodos*, 6(21-22) :45–68, 1954.

[61] LA Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24 :1337–1342, 1963.

[62] J Matyas. Random optimization. *Automation and Remote control*, 26(2) :246–253, 1965.

[63] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

[64] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2) :291–307, 1970.

[65] John H Holland. Adaptation in natural and artificial systems. *The University of Michigan Press*, 1 :975, 1975.

[66] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1) :156–166, 1977.

[67] Robert E Mercer and JR Sampson. Adaptive search using a reproductive meta-plan. *Kybernetes*, 7(3) :215–228, 1978.

[68] J Doyne Farmer, Norman H Packard, and Alan S Perelson. The immune system, adaptation, and machine learning. *Physica D : Nonlinear Phenomena*, 22(1-3) :187–204, 1986.

[69] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826 :1989, 1989.

[70] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization : Formulationdiscussion and generalization. In *Icga*, volume 93, pages 416–423. Citeseer, 1993.

[71] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2) :126–140, 1994.

[72] Roberto Battiti and Giampietro Tecchiolli. Training neural nets with the reactive tabu search. *IEEE transactions on neural networks*, 6(5) :1185–1200, 1995.

[73] J Kennedy and R Eberhart. Particle swarm optimization (pso). In *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948, 1995.

[74] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4) :341–359, 1997.

[75] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new heuristic optimization algorithm : harmony search. *simulation*, 76(2) :60–68, 2001.

[76] Sunil Nakrani and Craig Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12(3-4) :223–240, 2004.

[77] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty., 2005.

[78] KN Krishnanand and Debasish Ghose. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 84–91. IEEE, 2005.

[79] Omid Bozorg Haddad, Abbas Afshar, and Miguel A Mariño. Honey-bees mating optimization (hbmo) algorithm : a new heuristic approach for water resources optimization. *water resources management*, 20(5) :661–680, 2006.

[80] Esmaeil Atashpaz-Gargari and Caro Lucas. Imperialist competitive algorithm : an algorithm for optimization inspired by imperialistic competition. In *2007 IEEE congress on evolutionary computation*, pages 4661–4667. IEEE, 2007.

[81] Hamed Shah Hosseini. Problem solving by intelligent water drops. In *2007 IEEE congress on evolutionary computation*, pages 3226–3231. IEEE, 2007.

[82] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[83] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 210–214. IEEE, 2009.

[84] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. Gsa : a gravitational search algorithm. *Information sciences*, 179(13) :2232–2248, 2009.

[85] Ali Husseinzadeh Kashan. League championship algorithm : a new algorithm for numerical function optimization. In *2009 International Conference of Soft Computing and Pattern Recognition*, pages 43–48. IEEE, 2009.

[86] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

[87] Hamed Shah-Hosseini. Principal components analysis by the galaxy-based search algorithm : a novel metaheuristic for continuous optimisation. *International Journal of Computational Science and Engineering*, 6(1-2) :132–140, 2011.

[88] Kenichi Tamura and Keiichiro Yasuda. Spiral dynamics inspired optimization. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 15(8) :1116–1122, 2011.

[89] Kenichi Tamura and Keiichiro Yasuda. Primary study of spiral dynamics inspired optimization. *IEEJ Transactions on Electrical and Electronic Engineering*, 6(S1) :S98–S100, 2011.

[90] SONAL Rai, SK Mishra, and MUKESH Dubey. Teacher learning based optimization of assignment model. *Int. J. Mech. Prod. Eng. Res. Dev*, 3(5) :61–72, 2013.

[91] Amir Hossein Gandomi and Amir Hossein Alavi. Krill herd : a new bio-inspired optimization algorithm. *Communications in nonlinear science and numerical simulation*, 17(12) :4831–4845, 2012.

[92] S Salcedo-Sanz, J Del Ser, I Landa-Torres, S Gil-López, and A Portilla-Figueras. The coral reefs optimization algorithm : an efficient meta-heuristic for solving hard optimization problems. In *Proceedings of the 15th International Conference on Applied Stochastic Models and Data Analysis (ASMDA2013), Mataró*, pages 751–758, 2013.

[93] Mehdi Neshat, Ghodrat Sepidnam, and Mehdi Sargolzaei. Swallow swarm optimization algorithm : a new method to optimization. *Neural Computing and Applications*, 23(2) :429–454, 2013.

[94] Amir H Gandomi. Interior search algorithm (isa) : A novel approach for global optimization. *ISA Transactions*, 53(4) :1168–1183, 2014.

[95] RJ Kuo and Ferani E Zulvia. The gradient evolution algorithm : A new metaheuristic. *Information Sciences*, 316 :246–265, 2015.

[96] Maziar Yazdani and Fariborz Jolai. Lion optimization algorithm (loa) : a nature-inspired metaheuristic algorithm. *Journal of computational design and engineering*, 3(1) :24–36, 2016.

[97] Ali Husseinzadeh Kashan. A new metaheuristic for optimization : optics inspired optimization (oio). *Computers & Operations Research*, 55 :99–125, 2015.

[98] Yu-Jun Zheng. Water wave optimization : a new nature-inspired metaheuristic. *Computers & Operations Research*, 55 :1–11, 2015.

[99] Totok Ruki Biyanto, Henokh Yernias Fibrianto, Gunawan Nugroho, Agus Muhamad Hatta, Erny Listijorini, Titik Budiati, and Hairul Huda. Duelist algorithm : an algorithm inspired by how duelist improve their capabilities in a duel. In *International Conference on Swarm Intelligence*, pages 39–47. Springer, 2016.

[100] Totok R Biyanto, Sonny Irawan, Henokh Y Febrianto, Naindar Afdanny, Ahmad H Rahman, Kevin S Gunawan, Januar AD Pratama, Titania N Bethiana, et al. Killer whale algorithm : an algorithm inspired by the life of killer whale. *Procedia Computer Science*, 124 :151–157, 2017.

[101] Totok R Biyanto. Rain water optinization algorithm : NewtonŠs law of rain water movements. 2017.

[102] Ahmad Wedyan, Jacqueline Whalley, and Ajit Narayanan. Hydrological cycle algorithm for continuous optimization problems. *Journal of Optimization*, 2017, 2017.

[103] Nicholas Metropolis, Rosenbluth Arianna W, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equations of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6) :1087–1091, 1953.

[104] Dan Simon. Biogeography-based optimization. *IEEE transactions on evolutionary computation*, 12(6) :702–713, 2008.

[105] A Kaveh and S Talatahari. A novel heuristic optimization method : charged system search. *Acta Mechanica*, 213(3-4) :267–289, 2010.

[106] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. Gsa : a gravitational search algorithm. *Information sciences*, 179(13) :2232–2248, 2009.

[107] David E GOLDBERG. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[108] James Kennedy and Russell Eberhart. Particle swarm optimization. volume 4, pages 1942–1948, 1995.

[109] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1) :67–82, 1997.

[110] Christian Blum, Andrea Roli, and Michael Sampels. *Hybrid metaheuristics : an emerging approach to optimization*, volume 114. Springer, 2008.

[111] Laetitia Jourdan, Matthieu Basseur, and E-G Talbi. Hybridizing exact methods and metaheuristics : A taxonomy. *European Journal of Operational Research*, 199(3) :620–629, 2009.

[112] Olivier C Martin and Steve W Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63(1) :57–75, 1996.

[113] Thomas Stutzle and Holger Hoos. Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314. IEEE, 1997.

[114] Carlos Cotta and José M Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18(2) :137–153, 2003.

[115] Feng-Tse Lin, Cheng-Yan Kao, and Ching-Chi Hsu. Incorporating genetic algorithms into simulated annealing. In *Proceedings. International Symposium on Artificial Intelligence*, pages 290–297, 1991.

[116] V Nwana, Ken Darby-Dowman, and Gautam Mitra. A co-operative parallel heuristic for mixed zero–one linear programming : Combining simulated annealing with branch and bound. *European journal of operational research*, 164(1) :12–23, 2005.

[117] George A Croes. A method for solving traveling salesman problems. *Operations Research*, 6(6) :791–812, 1958.

[118] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164, 1985.

[119] Eneko Osaba, Enrique Onieva, Roberto Carballedo, Fernando Diaz, and Asier Perallos. An adaptive multi-crossover population algorithm for solving routing problems. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, pages 113–124. Springer, 2014.

[120] Bao Lin Lin, Xiaoyan Sun, and Sana Salous. Solving travelling salesman problem with an improved hybrid genetic algorithm. *Journal of computer and communications.*, 4(15) :98–106, 2016.

[121] Xiang Wang and Guoyi Xu. Hybrid differential evolution algorithm for traveling salesman problem. *Procedia Engineering*, 15 :2716–2720, 2011.

[122] Huiling Bao. A two-phase hybrid optimization algorithm for solving complex optimization problems. *Int J Smart Home*, 9(10) :27–36, 2015.

[123] Mesut Gündüz, Mustafa Servet Kiran, and Eren Özceylan. A hierarchic approach based on swarm intelligence to solve the traveling salesman problem. *Turkish Journal of Electrical Engineering & Computer Sciences*, 23(1) :103–117, 2015.

[124] Syed Basit Ali Bukhari, Aftab Ahmad, Syed Auon Raza, and Muhammad Noman Siddique. A ring crossover genetic algorithm for the unit commitment problem. *Turkish Journal of Electrical Engineering & Computer Sciences*, 24(5) :3862–3876, 2016.

[125] AE Eiben and Cees HM Van Kemenade. Diagonal crossover in genetic algorithms for numerical optimization. *Control and Cybernetics*, 26 :447–466, 1997.

[126] J Yoshida, M Miki, and Y Sakata. Best combinatorial crossover in genetic algorithms. *Joho Shori Gakkai Kenkyu Hokoku*, 85 :41–44, 2000.

[127] Aizhen Liu, Jiazhen Wang, Guodong Han, Suzhen Wang, and Jiafu Wen. Improved simulated annealing algorithm solving for 0/1 knapsack problem. In *Sixth International Conference on Intelligent Systems Design and Applications*, volume 2, pages 1159–1164. IEEE, 2006.

[128] Maya Hristakeva and Dipti Shrestha. Solving the 0-1 knapsack problem with genetic algorithms. In *Midwest instruction and computing symposium*, 2004.

[129] Ling He and Yanyan Huang. Research of ant colony algorithm and the application of 0–1 knapsack. In *2011 6th International Conference on Computer Science & Education (ICCSE)*, pages 464–467. IEEE, 2011.

[130] Kuk-Hyun Han and Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE transactions on evolutionary computation*, 6(6) :580–593, 2002.

[131] Yan Liu and Chao Liu. A schema-guiding evolutionary algorithm for 0-1 knapsack problem. In *2009 International Association of Computer Science and Information Technology-Spring Conference*, pages 160–164. IEEE, 2009.

[132] Dexuan Zou, Liqun Gao, Steven Li, and Jianhua Wu. Solving 0–1 knapsack problem by a novel global harmony search algorithm. *Applied Soft Computing*, 11(2) :1556–1564, 2011.

[133] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4) :568–581, 1964.

[134] Billy E Gillett and Leland R Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2) :340–349, 1974.

[135] Ann Melissa Campbell and Martin Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation science*, 38(3) :369–378, 2004.

[136] David M Ryan, Curt Hjorring, and Fred Glover. Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, 44(3) :289–296, 1993.

[137] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12) :1985–2002, 2004.

[138] Abel García-Nájera, John A Bullinaria, and Miguel A Gutiérrez-Andrade. An evolutionary approach for multi-objective vehicle routing problems with backhauls. *Computers & Industrial Engineering*, 81 :90–108, 2015.

[139] RaúL BañOs, Julio Ortega, ConsolacióN Gil, Antonio L MáRquez, and Francisco De Toro. A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & Industrial Engineering*, 65(2) :286–296, 2013.

[140] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. An evolutionary algorithm for the vehicle routing problem with route balancing. *European Journal of Operational Research*, 195(3) :761–769, 2009.

[141] Shuguang Liu, Weilai Huang, and Huiming Ma. An effective genetic algorithm for the fleet size and mix vehicle routing problems. *Transportation Research Part E : Logistics and Transportation Review*, 45(3) :434–445, 2009.

[142] Jorge E Mendoza, Andrés L Medaglia, and Nubia Velasco. An evolutionary-based decision support system for vehicle routing : The case of a public utility. *Decision Support Systems*, 46(3) :730–742, 2009.

[143] Keivan Ghoseiri and Seyed Farid Ghannadpour. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing*, 10(4) :1096–1107, 2010.

[144] Yong Shi, Toufik Boudouh, and Olivier Grunder. A hybrid genetic algorithm for a home health care routing problem with time window and fuzzy demand. *Expert Systems with Applications*, 72 :160–176, 2017.

[145] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4) :421–451, 1993.

[146] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10) :1276–1290, 1994.

[147] Éric Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8) :661–673, 1993.

[148] Yves Rochat and Éric D Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1) :147–167, 1995.

[149] Jiefeng Xu and James P Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30(4) :379–393, 1996.

[150] César Rego and Catherine Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In *Meta-Heuristics*, pages 661–675. Springer, 1996.

[151] César Rego. A subpath ejection method for the vehicle routing problem. *Management Science*, 44(10) :1447–1459, 1998.

[152] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4) :333–346, 2003.

[153] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8) :928–936, 2001.

[154] Victor R Vidal V Werra D Silver, E. A. A tutorial on heuristic methods. *European Journal of Operational Research*, 5 :153–162, 1980.

[155] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3) :268–308, 2003.

[156] Eduardo Rodriguez-Tello, Jin-Kao Hao, and Jose Torres-Jimenez. A comparison of memetic recombination operators for the minla problem. In *Mexican International Conference on Artificial Intelligence*, pages 613–622. Springer, 2005.

[157] Eneko Osaba and Fernando Díaz. Comparison of a memetic algorithm and a tabu search algorithm for the traveling salesman problem. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 131–136. IEEE, 2012.

[158] Shangce Gao, Wei Wang, Hongwei Dai, Fangjia Li, and Zheng Tang. Improved clonal selection algorithm combined with ant colony optimization. *IEICE transactions on information and systems*, 91(6) :1813–1823, 2008.

[159] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. In *Handbook of metaheuristics*, pages 287–319. Springer, 2003.

[160] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization : A survey. *Applied Soft Computing*, 11(6) :4135–4151, 2011.

[161] Ahmed Fouad Ali and Aboul-Ella Hassanien. A survey of metaheuristics methods for bioinformatics applications. In *Applications of Intelligent Optimization in Biology and Medicine*, pages 23–46. Springer, 2016.

[162] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem : a computational study*. Princeton university press, 2006.

[163] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Metaheuristics for hard optimization : methods and case studies*. Springer Science & Business Media, 2006.

[164] Holger H Hoos and Thomas Stützle. *Stochastic local search : Foundations and applications*. Elsevier, 2004.

[165] Sun-Chong Wang. *Interdisciplinary computing in Java programming*, volume 743. Springer Science & Business Media, 2012.

[166] Stefan Edelkamp and Stefan Schroedl. *Heuristic search : theory and applications*. Elsevier, 2011.

[167] Patrick Siarry and Zbigniew Michalewicz. *Advances in metaheuristics for hard optimization*. Springer Science & Business Media, 2007.

[168] Roman M Krzanowski and Jonathan Raper. *Spatial evolutionary modeling*. Oxford University Press, 2001.

[169] Robert Ghanea-Hercock. *Applied evolutionary algorithms in Java*. Springer Science & Business Media, 2013.

[170] Lee Jacobson and Burak Kanber. *Genetic algorithms in Java basics*. Springer, 2015.

[171] Erik D Goodman. Introduction to genetic algorithms. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 205–226. ACM, 2014.

[172] Patrick Siarry, Lhassane Idoumghar, and Julien Lepagnot. *Swarm Intelligence Based Optimization*. Springer, 2016.

[173] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[174] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2) :95–99, 1988.

[175] Patrick Siarry. *Metaheuristics*. Springer, 2016.

[176] Luca Maria Gambardella and Marco Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3) :237–255, 2000.

[177] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7 :225–330, 1995.